AFRL-IF-RS-TR-2005-354
Final Technical Report
October 2005

INTELLIGENT SECURITY CONSOLE ARCHITECTURE

University of Memphis

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J807

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-354 has been reviewed and is approved for publication

APPROVED: /s/

ROGER J. DZIEGIEL, JR.
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	OCTOBER 2005	Final Mar 00 – Mar 05

4. TITLE AND SUBTITLE
INTELLIGENT SECURITY CONSOLE ARCHITECTURE

5. FUNDING NUMBERS
C - F30602-00-2-0514
PE - 62702E
PR - IAST
TA - 00
WU - 18

6. AUTHOR(S)
Dipankar Dasgupta

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Intelligent Security Systems Research Laboratory
The University of Memphis
Memphis Tennessee 38152

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency AFRL/IFED
3701 North Fairfax Drive 525 Brooks Road
Arlington Virginia 22203-1714 Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-354

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Roger J. Dziegiel, Jr./IFED/(315) 330-2185/ Roger.Dziegiel@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
This report addresses the design of an Intelligent Security Console equipped with Intrusion Detection Message Exchange Format (IDMEF) Objects' data mining for the DARPA Ultra*Log Program. It supports the scalable Monitoring and Response security console architecture. The Data Mining capability requires scalability of message management, that has been ensured through incorporation of an XML Database (eXist). Security console is used to query for IDMEF alerts generated across the society by various sensors (including COTS). The results are shown as a tree with the structure corresponding to the security communities' hierarchy in getting the society status through queries and alert messages. The latest version (4.1) of the security console is designed to mine frequent patterns in Intrusion attacks with an XML repository for collecting and organizing alerts and event messages. This ensures scalability and organized storage of voluminous information over a period of time.

14. SUBJECT TERMS
Agents, Intrusion Detection, Security Console

15. NUMBER OF PAGES
117

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1 Executive Summary

1.1 Purpose

Proposed Software Title: Intelligent Security Console

The University of Memphis effort for FY 2004 includes the design of an Intelligent Security Console equipped with IDMEF Objects' data mining. It supports the scalable Monitoring and Response security console architecture. The Data Mining capability requires scalability of message management, that has been ensured through incorporation of an XML Database (eXist). In this document, we present the architectural design and highlight the issues being addressed.

Security console is used to query for IDMEF alerts generated across the society by various sensors (including COTS). The results are shown as a tree with the structure corresponding to the security communities' hierarchy in getting the society status through queries and alert messages. The latest version (4.1) of the security console is designed to mine frequent patterns in Intrusion attacks with an XML repository for collecting and organizing alerts and event messages. This ensures scalability and organized storage of voluminous information over a period of time.

The important features of the security console are as follows:

- Allow defining queries using the Drilldown architecture. The security console provides a GUI component that allows editing, storing and retrieving of queries and showing the results in a tree form.

- To provide the user a query-builder tool to define queries in an easy way. Examples of these tools include predefined queries and query by example. These tools allow the user to generate queries with little effort and without much knowledge of the specific query format. However, the users will be able to modify the generated query to adapt it to their needs.

- Supports the actions required for the current Drill Down architecture. Two different queries can be sent. One is the root query with the predicate and the other type of query is an expansion query, which has only the location of the expansion and no predicate.

- Implements a high level communication protocol for sending queries to a security console manager and for retrieving results from them to present in the Security Console (SC) client. This protocol encapsulates a HTTPS protocol (via servlets) and uses XML documents for the information transfer. Finally, it implements a *Keep-Alive* mechanism for regular updates.

- To display query results showing summary information, like Create Time, Detect Time, Security Manager, Analyzer, Source, Target, Node .

- To allow the user to sort query results based on different output fields (such as date, event-type, alert-level, etc.).

- To display individual IDMEF message details. The security console will allow the user to inspect the details of the messages using either of two primary views: textual view (XML representation of the message) and tree view (JTree representation of the message.

- Full Featured XML database support for managing the IDMEF messages as Collections, over which queries ('XQuery') could be made.

- Special Views for message statistics using graphical Charts and tree.

- Enhanced Structures to represent Mining Episode Results as Hierarchical tree and Graph. The Generated Rules can be visualized using a List Structure as well as 3D Graphics.

- **Scalability:** Multiple Security Console (SC) managers and multiple SC clients may be used. The SC client has to know only one manager's address to get the information of all the other managers in the society. Also each manager allows connections from multiple clients and maintains a map between users and their queries.

  Managing messages for off-line Mining is done using a database repository that has support for Insertion, Deletion and Updating. The repository is equipped to store hierarchical messages over a period of time.

- **Robustness:** The Security console is designed to be robust to failures. For instance if a SC client crashes and reloads again, the SC client is updated with all its queries and results. Also each SC manager uses the persistence mechanism to recover after the crash. The SC Mining component has the ability to deal with voluminous XML messages for Statistics Analysis and Episode mining. This is made possible by grouping specific numbers and them aggregating them in a step-wise manner.

- **Mining XML Messages:** The data mining features offer specialized queries for doing statistical analysis of XML messages and apply frequent episodes learning to generate patterns of high interest to the user. These help indicate patterns of attack through valuable statistics.

1.2 Scope

Tasks done till FY 2004:

- A client-manager protocol that allows multiple clients and multiple SC managers to communicate among each other. The communication is done with xml. Communication objects represent different commands and results. Some commands from the SC client are generated based on the user actions, while others are designed for the communication protocol and robustness support.

- Multiple SC managers are allowed in the society, which means multiple clients can connect to the same SC manager and send queries and receive results simultaneously. The SC manager maintains a map of users to queries (and results). Typically one SC manager is inserted per community.

- The client-manager protocol is designed keeping in view robustness issues like, delay in communication; SC client failure, SC manager failure and communication failure. Client has no persistence; all the information is stored in the SC manager. If a SC client crashes and is restarted (possibly in a different computer), it sends a get-all command to update itself. The SC manager sends all the results for the queries to the restarted SC client. Note that even if a SC client crashes it still has entry in the map in the SC manager.

- The robustness issue has been tested with the previous aggagent architecture, with the new drill down architecture.

- Now in the security console the result of query can be a simple consolidated event at the root level or fully expanded tree until the leaves of the security communities' hierarchy where, the leaves are the sensor agents.

- According to the architecture of the current implementation when a query is sent the first time from the SC client the SC manager makes a "DrillDownQuery" and sends it to the appropriate MnRManager and listens for the results. The results are sent to the SC client as and when available which are consolidated events.

- The user at the SC client may choose to expand at any particular point (the root to start with) by sending an expansion command. The SC manager now sends a "DetailsDrillDownQuery" to the appropriate MnRManager. The results of this query can be consolidated as well as absolute events at the sensor level. Note that a SC manager at any level may have sensors registered to it.

- Again the SC manager listens to the results and sends the results and updates to the SC client, as and when available. Also the consolidated events sent may further be expanded and the process can be continued until the whole hierarchy has been expanded.

- SC Client follows the communication protocol to send the commands to the SC manager and receive the results. Also SC client sends the get-all command the first time after it is started.

- GUI allows the user to choose any SC manager available in the society. Also the user may choose a destination community represented by an MnRManger for a drill down query. The user may send expansions only after receiving consolidated events. The received results as shown as a tree. The user may expand by double clicking at a particular point. Expansions may be done only at the MnRManager level. Expansions at the sensor agent level are invalid. To guide help the user in this aspect, the GUI shows the MnRManagers and Sensor agents in different colors.

- The complete functional Mining tool is now integrated into the SC. The basic functionality of this tool is to find the occurrence of frequent events and analyze their behavior in the form of association rules and related episodes of various lengths. This can also be used as a recommendation mechanism for the Administrative user that enables him/her to scrutinize the Society events in a better way.

- The XML messages that are mined can be done in two ways – Serial or Parallel. The mined episodes are further analyzed to find relationship among them. This, results in the generation of Rules that are indications of the behavior pattern among the messages of various events generated in the Society.

- The mining results, their representations and their visualization have been made as graphical as possible. Detailed statistics and relationships are depicted in a graphical form involving time sequence.

- The Data Base repository, namely eXist, has been used for storing and record-keeping purposes. The database is fully supportive of the XML messages and has a suitable structure called collections to store the event messages in a hierarchical manner. The database further supports maintenance by incorporating provisions for update, modify, delete and query any record(s) in the collections' structure.

- The Data Base is fully supportive of the XQuery language that can directly deal with querying the XML messages. SC provides the user with the GUI to interface with the database via the *dbDriver*. Commands to manipulate the database records are inbuilt into the SC.

- SC collects the messages for off-line mining in two distinct ways. One, the Results can be directly pushed into the database that creates the collections structure. Other, the results (Alert XML messages) could be logged on to a file. This file is saved in the machine and can later be uploaded to the database.

- The database administrator can do a backup of the eXist database. For this, the administrator must save the files inside a directory. If the files are not there, it means that library will start from scratch (the library will create the files). Note that by default this directory is in **<SC install directory>/eXist/webapp/WEB-INF/data**, but the administrator can move files to other directory.

- SC runs with the embedded eXist XML database. But SC can also run with a stand alone eXist XML database. In the stand alone mode, the database can run in the same machine where SC is running or can run in other machine. Indeed, a centralized eXist DB can be the general server for all the IDEMF messages coming from different SCs.

Plan for Future Enhancement:

- The idea is to have two role-based SC managers. The SC manager can take the role of an enclave-console-manager or society-console-manager. We can have one enclave-console-manager per security enclave and one society-console-manager per society. We may choose to have more than one to avoid single point of failure.
- Society Console Manager is responsible to provide the SC client with the information about the society. The SC client expects the following information from this SC manager:

    - List of SC managers up and running along with their locations.

    - List of all the MnRManagers representing the different security enclaves in the society.

- The SC client sends commands to the society console manager to get information about the society. This information is then provided to the GUI to let the user choose from a list. The user can choose the SC manager to connect to from a list. Also the user can choose the target manager to which a query has to be sent from a list. This avoids the necessity of knowing the name of the destination manager and also avoids sending the queries of a SC manager that does not exist.

- The client-manger protocol has to be update and extended to support the above communication.

- The console manger should make sure that all the connections to the SC clients are restored and the sending-results are continued after rehydration.

- Get-all mechanism to work with the new drill down architecture. A new technique has to be followed to make sure the SC client gets all the information lost as a result of its crash, which may be a new CmrRelay to the MnRManager.

- When an SC client is started, the user is asked for authentication. This user authentication is used to identify the user uniquely. So the map of users and the queries in the SC manager is done using these user ids. These ids are obtained through the http connection. This functionality has yet to be implemented. As of now, the GUI allows the user to specify the user id and authentication is not used. This is a key feature for the robustness of the security console.

- Extensive testing is required in the following aspects:

  -Robustness

    - SC Client failure
    - SC Manager failure
    - Communication failure

  - Scalability

    - Multiple clients, single SC manager
    - Single client, multiple managers

  -Drill Down Mechanism

  -Role based SC manager

    - Society console manager
    - Enclave console manager

  -Graphical user interface.

- Continued support to the submitted versions of security console and the cots (snort and tripwire) plug-ins.

1.3 Design Overview

We developed a security console for reporting M&R outcomes by means of four different components: GUI, Query module, and Communication component.

- **GUI Component:** This component is composed of a Main Interface and Visualization Tools. Main Interface exhibits the existence of various components and facilities the user can make use of. This interface also has 'Helper GUIs' which helps the user in creating, editing and storing queries. Visualization Tools portray different views, which are visualizations of IDMEF objects. These tools include certain powerful graphic libraries for showing the inter-relationships among the related events, statistics and rules generated.

- **Query Component:** This module is responsible for receiving the query sentences from the GUI module, sending the associated query commands to the M&R manager through the communication component, storing the query results sends back for the M&R manager through the communication module, and applying the desired filtering tools over such stored query results.

- **Communication Component:** This component performs the message exchanging process (query commands and IDMEF objects), between the servlets in the M&R Manager and the SC client in the security console, and the translation of this messages into the appropriated structure, for example, translating a XML document to the appropriate query results collection.

- **Security Console Manager:** The Security Console Manager is responsible to receive and execute commands from the security console client. The SC manager supports commands like, publish query, delete query, expand query, get all query results, and get society information among others. The SC manager also sends the results of queries and the results of other commands to the SC client. Several clients can connect to the SC manager and each SC client is identified by the user id. The SC manager groups all the queries by user.

  The SC manager keeps the information of the users even after they disconnect. So when the user connects again, possibly after a crash, the SC manager can update the SC client with all the results that are lost. Blackboard persistence is used for crash recovery.

- **Database Management Component:** This component assists the user in managing the messages in an XML repository over a period of time in a structured fashion. Normal commands for managing like Insert, Delete and Update are incorporated along with the user-friendly View for showing the top-level organization of 'Collections'. Here the database is a generic XML database for the storage of IDMEF objects that are filtered results for the Agg-queries sent to the Console Manager, in a predefined structure. Duplicate and redundant data are filtered out by examining the <ident> tag in the incoming messages.
  The messages are deleted from the collection based upon the *date, time, SCMID.* The user has the option to select any range of records to extract and analyze interesting patterns.

- **XML Query Component:** The Query language 'XQuery' is used to extract and aggregate qualitative information from the XML database repository. The resulting XML result is processed as the DOM document and DOM (Document Object Model) API (Application Programmer's Interface) is utilized to convert the data into objects collection for two major tasks.

1. Statistical Visualization. This objects collection form a specific dataset as per the query parameters that can be effectively visualized using graphical Bar Chart and time Sequence chart.

2. Episode Mining: The objects collection contains events of interest that qualify a certain threshold and become candidates of Frequent Episodes. These are then associated based on a specific heuristic to get knowledge of rules interlinking these frequent occurrences.

- **Mining Component:** *Knowledge Discovery task*: To better analyze and determine the events patterns, we made use of the Frequent Episodes and Association Rule Mining to generate the candidate episodes and event sequence based on a set of Observed parameters, like Confidence, Frequency, Window Size, Length and Mining type (Serial or Parallel sequence). Given a class of episodes and an input sequence of events, we find all sequential episodes that occur frequently in the event sequence. This classification mechanism helps to group related patterns amongst events together.

  Association Rules and Estimation: Given all frequent episodes, we then apply heuristics to generate relations among frequent candidates (of events) and identify associations that forms rules on the basis of their relative confidences.

2 Functional Description

2.1 Introduction

The Intelligent M&R Security console is a standalone application that enables the user to interact with the M&R manager. The information produced by M&R sensors and analyzers about events and alerts is represented as IDMEF objects. The user can browse these event objects through the M&R security console. Accordingly, the M&R security console should provide appropriate tools that make the tasks of searching and visualizing security thread messages more efficient. In particular, query results can be expanded or deleted as necessary.

The query results contain many events of interest that have some associations among themselves. The Database management and Mining component have the capability to mine the incoming messages off-line from a repository and visualize them effectively. For this, the incoming messages are stored into the repository either at run-time or from a logged file (that collects messages at real-time) at later time. This ensures maintainability and manageability.

The important features of Security Console (SC) and the Intelligent SC M&R manager will be as follows:

2.1.1 Security Console Client:

Allow defining aggregation queries for IDMEF objects. The security console will provide a GUI component that allows editing, storing, retrieving queries, analyze results and recommend association rules.

To provide the user a *query-builder tool* to define queries in an easy way. Examples of these tools include *predefined queries* and *query by example*. These tools allow the user to generate queries with little effort and without knowledge of the specific query format. However, the users will be able to modify the generated query to adapt it to their needs.

- Use the HTTP protocol to communicate with the SC M&R manager via servlet to send queries and receive query results.

- To display query results showing summary information, like creation time, analyzer, type (heartbeat or alert) etc.

- To allow the user to sort query results based on different output fields (such as date, event-type, alert-level, etc.).

- Provide user-defined filters to refine query results. The user will be able to apply new queries over the query results. This generates precise query results and the process can be iterated as many times as the user wants.

- To display individual IDMEF message details. The security console will allow the user to inspect the details of the messages using either of two primary views: textual view (XML representation of the message) and tree view (JTree representation of message structure).

• To display IDMEF messages using specialized views. An example of this kind of view is a *Time Series View*: this view displays the variables of a set of Heartbeat messages using a graphical representation.

• Use the aggregation mechanism for the execution of queries.

2.1.2 Security Console Manager

It provides these basic functionalities.

• To receive queries and commands from security console.

• To update a reconnected SC Client with all the lost query results.

• Support Drilldown mechanism for consolidated and normal events.

• Uses keep-alive mechanism to send results to the security console as and when they are available.

• Support for robustness on the SC manager side using blackboard persistence.

• Facility to update a restarted client (after crash) with queries and results thus supporting client robustness.

2.2 System Capabilities

2.2.5.2 Security Console Client

GUI component helps the user in creating queries related to Heartbeat and Alert objects. These queries can be edited, and can be saved. These queries can then be executed and results are displayed as a list. Selected list of results can be viewed in different **Views**: Tree View, Text View are provided by default. Other views can also be plugged in to the console.

For the Mining Module, the User is provided with views in the form of time-series chart, bar chart and 3-D graphics. The User has the option to save the Statistical query result, Frequent Episodes and Rules.

The functionality of the security console system will be implemented through three components: GUI Component, Query Component, Communication Component, Database Management Component and XML Query Component. Their capabilities are described as follows:

- GUI Component contains Main Interface and Visualization Tools, which provides the following functionalities:
    - Connect with different SC managers.
    - Display information of the available SC managers and M&R managers.

- An interface to specify the query with ‘Helper GUIs’ to help the user in creating, editing and storing queries.
- Display Query results showing summary information, providing the user with tools to filter and sort these results.
- Display IDMEF objects in different Views (Text, Tree, Time-Series View).
- Helping user in creating queries related to Heartbeat and Alert objects.
- Created queries can be edited, and can be saved.
- Queries are can be executed and results are shown in different view.
- The user can change the scope of the query specifying the capabilities.
- The user can manipulate the XML database having a collection of messages over a period of time. For this, a table, with a table sorter is provided that can change the relative orderings based upon the Date, Hour and QueryId of the messages.
- The User is provided with tree and Graphical view of the Query result over the messages selected in the Statistical Analysis panel.
- The mined frequent episodes have an associated Visualization showing the hierarchy as a tree and the message statistics as a graphical chart. Each individual episode’s events are displayed as a connected graph.
- The Mined Rules can be visualized in a Text as well as a 3-dimensional Graphical representation. These Rules can be sorted based upon certain parameters. Rules can be viewed in small groups or in its entirety.
- The GUI provides the option to save the individual message statistics, episodes and rules for further assessment if any.

- The Query component performs
  - Sending query sentences to M&R manager and receiving data through communication module.
  - Storing the query result IDMEF objects in a buffer.
  - Filtering the IDMEF objects based on different criteria.
- The Communication component performs
  - Low-level communication with M&R Manager.
  - Sending query sentences and receiving IDMEF objects in XML format.
  - Converting XML IDMEF objects to Java IDMEF objects.

- Sending converted IDMEF objects to the Query component.
- Saves a log of incoming messages into a file or directly to the database as desired.

- The Database management component performs
  - Initialization and low-level communication with the XML eXist database.
  - Sending commands for update, delete of collections in the database.
  - Data Communication with the database involving creating queries over a group of Database Collections and retrieving result using the Document Object Model.
  - Sending the retrieved result to the XML Query Component and Mining Component.

- The XML Query Component performs
  - Sending query predicates to the database and receiving data through the database management component.
  - Building the query results in a presentable format and sending it to GUI component or data mining component as appropriate.

2.2.5.2 Security Console Manager

The Security Console Manager receives queries and commands from the security console and sends results back. This agent has a servlet component and two plugins.

- Security Console Servlet Component: This servlet component accepts connections from new SC clients and sends/receives information to/form connected SC clients. The information sent/received in done through a channel layer object in the blackboard. This channel layer is used to abstract the details of the communication protocol. The servlet implements a keep-alive mechanism to simulate push operation. This is useful, as the SC manager has to send the results when available.

- Query Manager Plugin: Receives and processes commands sent from the SC client through the channel layer. This plugin is responsible to publish relays for new queries/expansions. The queries are hierarchical, that is, there is a root query and then there are expansions and expansions of expansions and so on. When a SC client reconnects after a crash the results of the queries at the root level are sent to the SC client by the query manager plugin. Also, this plugin looks for any queries one level below root queries and publishes new relays to get all the lost query results.

- Query Results Plugin: Sends the results of queries in the blackboard whenever available. The queries are grouped by the user-id. The results of the queries are sent to the appropriate user. When an SC client reconnects after a crash, this plugin sends the results of the queries at all the levels below the root level.

2.3 Design Approach

The basic infrastructure of the system is implemented using Java language. The management of SC Communication and event queries uses the Java IDMEF and Jython libraries. The management of database makes use of the eXist library and its queries have been designed using the XQuery language library. For graphic visualizations, chart2D, graph and java3D libraries have been utilized. To deal with message manipulation, the Document Object Model (DOM) library has been used extensively.

3 System Structure

3.1 Introduction

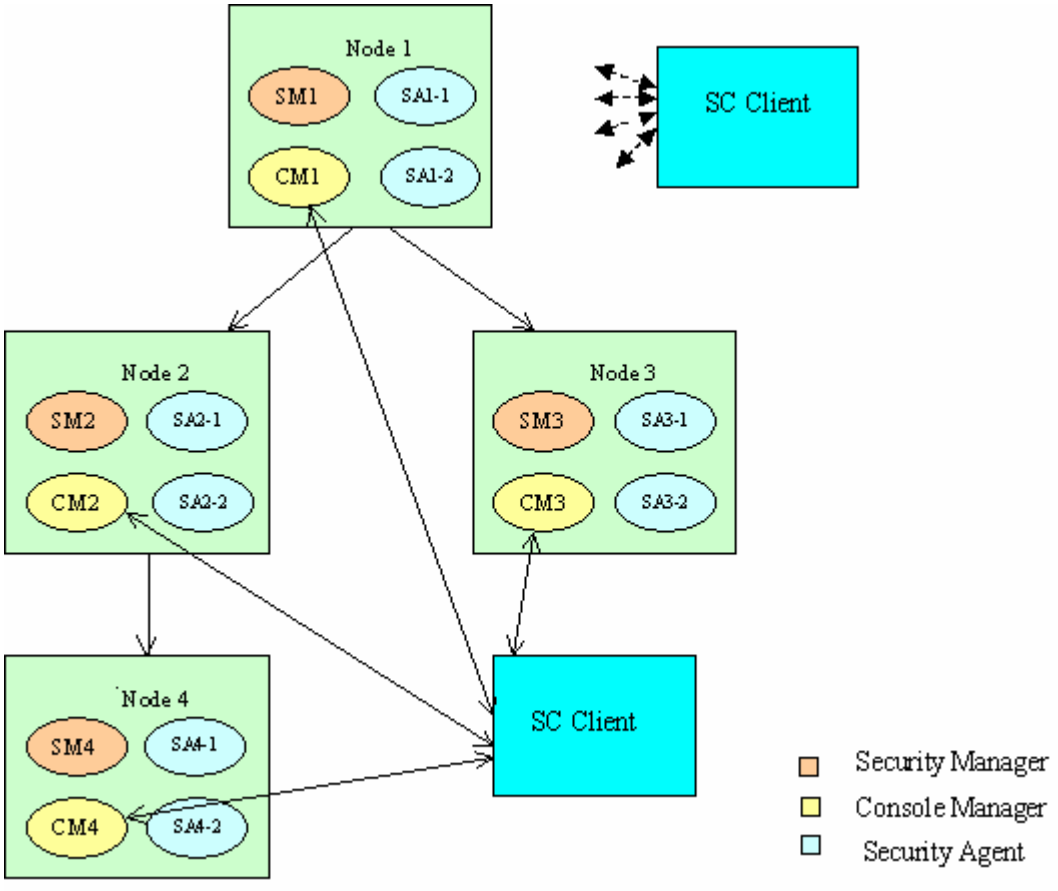


Figure 1. Security console managers and SC clients in a simple society

Security console is composed of security console client and security console manager. Typically we can have one security console manager per security enclave. There can be any number of SC clients. The figure shows a simple society where we can have one SC client and four-console manager each in a different enclave. The operation of each component in Security Console Client and the Security Console manager are described below:

3.2 Decomposition Description

3.2.1 Security Console Client Architecture

The security console system structure is shown in *Figure 2*.

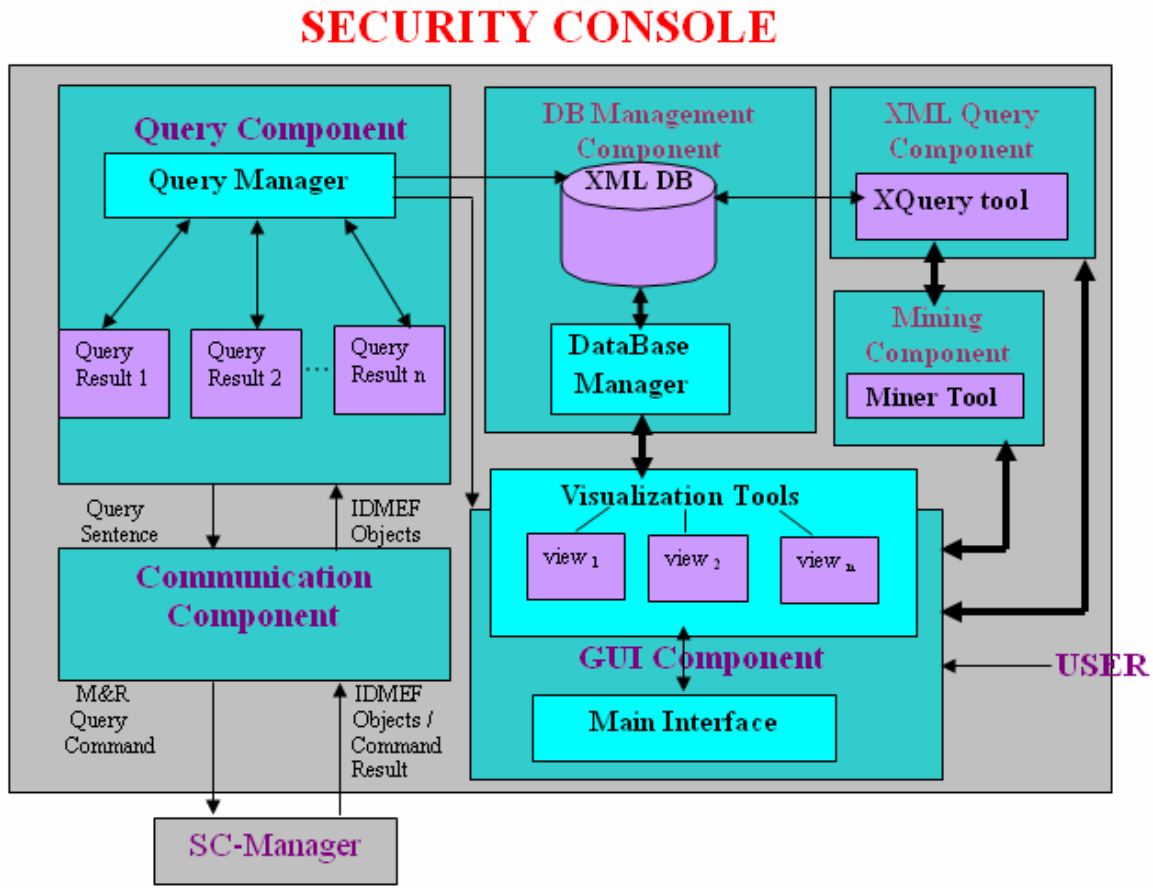


Figure 2. Security console Architecture

As mentioned earlier, the security console is composed of six different components: GUI, Query, Communication, DB Management, XML Query and Mining Component.

3.2.1.1 GUI Component:

The entire GUI is divided into small SnapIns, which can be plugged into the Main Application. Each SnapIn is unique and has specific task to perform. They are however interlinked with each other. The Mining SnapIn however can be run independently of other SnapIn.

- Setup SnapIn: Allows the user to connect to the Security Console Manager through http connection. This is the primary Snap In for getting Society data. The SC manager to connect to can be added, deleted and modified.

- QuerySnapIn: Using this SnapIn the user can create and execute a query and also visualize its results. It is divided as two main subcomponents:

    - QueryEditor: It allows the user to type a query and sends it to the society. The user can load predefined queries and save own queries to disk. The user can also refine a query result by applying a new query over it (filter).

- QueryBuilderSnapIn: Allows the user to build a query by specifying values for the attributes.

- Results SnapIn: It shows the retrieved IDMEF objects in a table. Set of objects can be selected and can be viewed in different views (TreeView, Text View, Time-Series View).

- Mining SnapIn: This SnapIn provides the basic User Interface for carrying out the entire knowledge mining over XML messages (that are in IDMEF format).

- Log SnapIn: Keeps logs of events.

3.2.1.2 Query Component

The query component is composed of the query manager module and a set of query results modules.

The query manager is responsible for:

- Retrieving query results from the society through the communication component

- Storing all the query results obtained by the queries performed by the security console operator.

- Storing all the urgent alerts (incoming alerts) that are generated by some Security component and send to the security console through the communication component.

- Filtering stored query results: Each query result module is responsible for keeping the result queries obtained by retrieving information from the M&R manager or by filtering previous query results.

There are basically two types of query result modules:

1. *Filtering query results*: These are obtained by applying a query-over-query i.e. filtering a query result that is stored in the query component. This type of query results does not establish communication with the M&R manager.

2. *Society query results*: These query results are obtained from the society when the security console operator requests it or when some M&R sensor published new urgent alerts in the M&R manager blackboard. The IDMEF objects are added when they are available. In this way a permanent communication with the KeepAlive servlet in the M&R Manager is established by the communication component. When the security console request some objects from the society, the associated query sentence is sent to the M&R manager by establishing a temporal connection with the Aggregation servlet in it.

   To identify the different query results obtained and stored in the query component, the security console:

   1. Sets the ***ids*** of the filtering query results,

   2. Uses the ***ids*** assigned by the aggregation infrastructure to the society query results, and

3. Uses a predefined ***id*** for the urgent alerts and for the society.

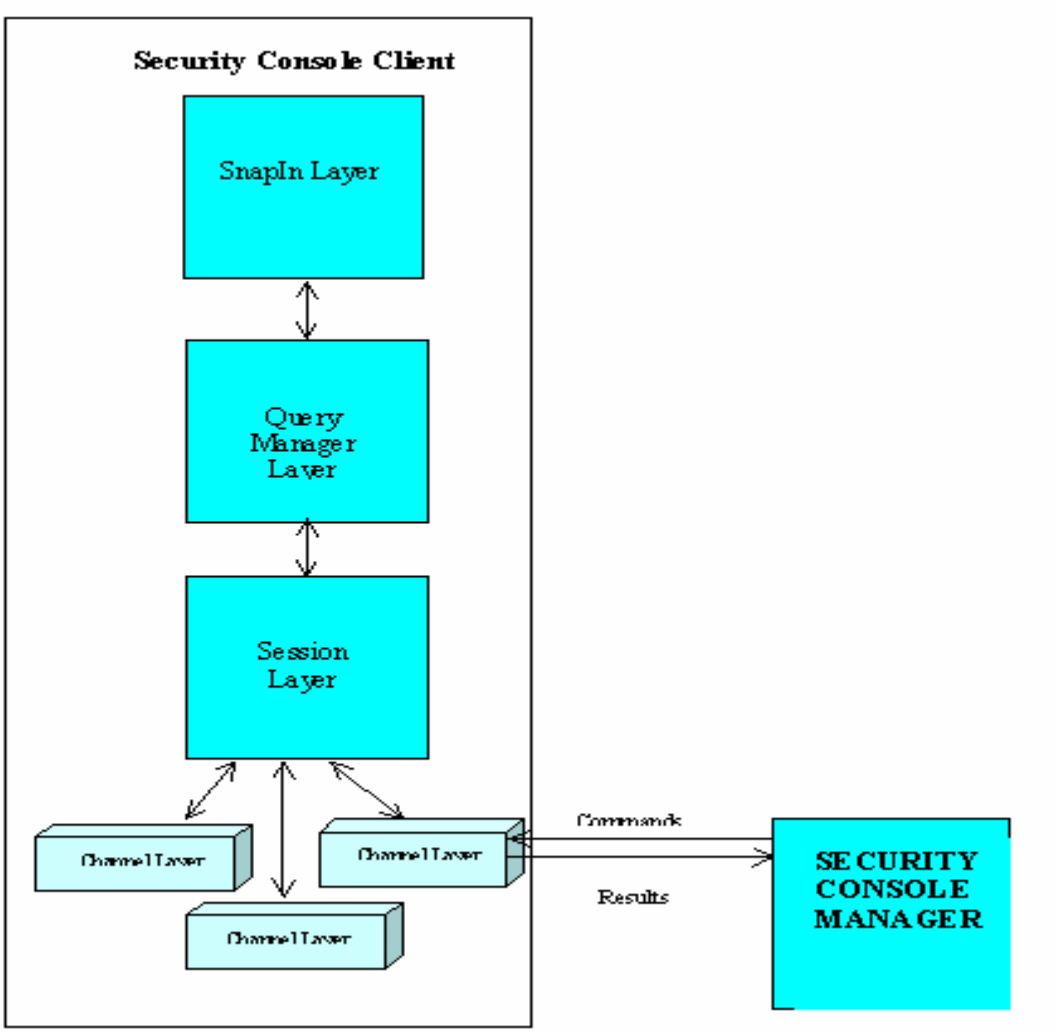


Figure *3 layers in the Communication Protocol (Client side)*

3.2.1.3 Communication Component:

This component is responsible for:

1. Establishing a communication channel between the security console and the M&R manager.
2. Converting a query sentence to a query command to be sent to the M&R manager
3. Sending the query command to the M&R manager
4. Receiving the XML IDMEF objects from the M&R manager
5. Converting the XML IDMEF object to Java IDMEF objects.

To send different M&R commands and to communicate with different SC managers at the same time, the communication component contains a synchronization mechanism between the SC client module and the servlet components in the SC managers. Also, the communication protocol is designed following a multi-layer communication scheme, see Figure 3. As shown, the upper layers should not be modified if the communication channel is modified.

3.2.1.4 Database Management Component:

This component is associated with an XML Database from 'eXist' and basically communicates on behalf of the user apart from the general initialization process. See Figure 2.

The database manager is responsible for:

1. Performing Update, Delete query over the database messages' collection.

2. Initializing the connection to the database via the provided driver.

3. Retrieving the high level view of the database organization for the collections', to the GUI component. The component is equipped with the capability to iterate through the XML collections structure in the database and sieve out the messages as appropriate.

The XML Query Component creates complex query structures and this component takes it as the input to process the messages' collection database and returns a Document Object Model, back.

3.2.1.5 XML Query Component:

This component use the XQuery tool for generating complex query structures for getting the specific result that is of interest.

The Component plays the following role:

1. On getting a specific query command from the GUI component it creates a XQuery predicate and sends it to the DB management component.

2. Sends the Result to the GUI component for Visualization.

3. Communicates with the mining component for creation of a XQuery predicate and send it to the DB management component for retrieving messages of interest.

4. Sends the Result to the Mining Component for further Processing.

All the basic data communication involves the exchange of data units as a javax DOM document that can be iterated and queried repeatedly by various modules as and when desired.

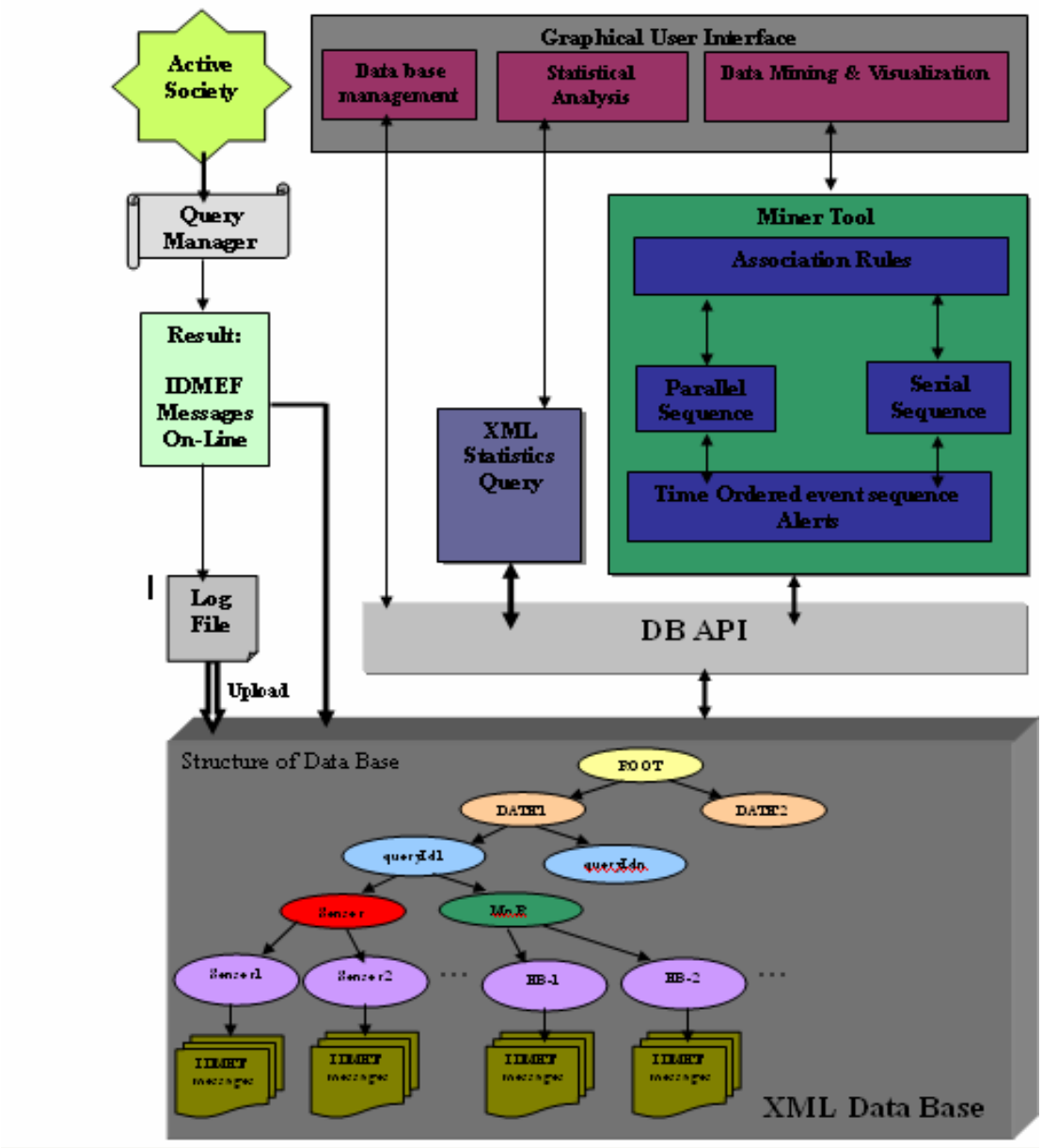


Figure 4. The Miner Tool of Mining Component, interacting with the Database.

3.2.1.6 Mining Component:

This Component is responsible for extracting useful information from the logged results. This information is then utilized to find frequent patterns among the events. The Statistics and frequent associations provide the user with important decision making (like specific attack patterns, controlling the network nodes) and monitoring. See Figure 4.

The mining is done off-line on the aggregated data collected over a period of time in the database. The Component has inbuilt features for generating suitable candidates and then help to perform good recommendation based upon the rules that it generates. All the parameters that it receives from the GUI Component are fed to it by the User.

The Mining Component basically performs the following tasks:

1. Running Algorithms for Parallel and Serial Sequence of events that are run over the Alert messages which are time ordered event sequences in IDMEF message format.

2. Sending the generated sequences to the GUI Component for Visualization.

3. Finds Rules that involves the events and gives it to the GUI component for Visualization and further analysis like recommendation and/or making important critical decisions.

4. For any Query processing it communicates with the XML Query Component.

5. Retrieving the high level view of the database organization for the collections', to the GUI component. The component is equipped with the capability to iterate through the XML collections structure in the database and sieve out the messages as appropriate.

3.2.2 Security Console Manager Architecture

The following Figure 5 shows the components in the Security Console Manager and its communication with the SC client and the MnR Manager.

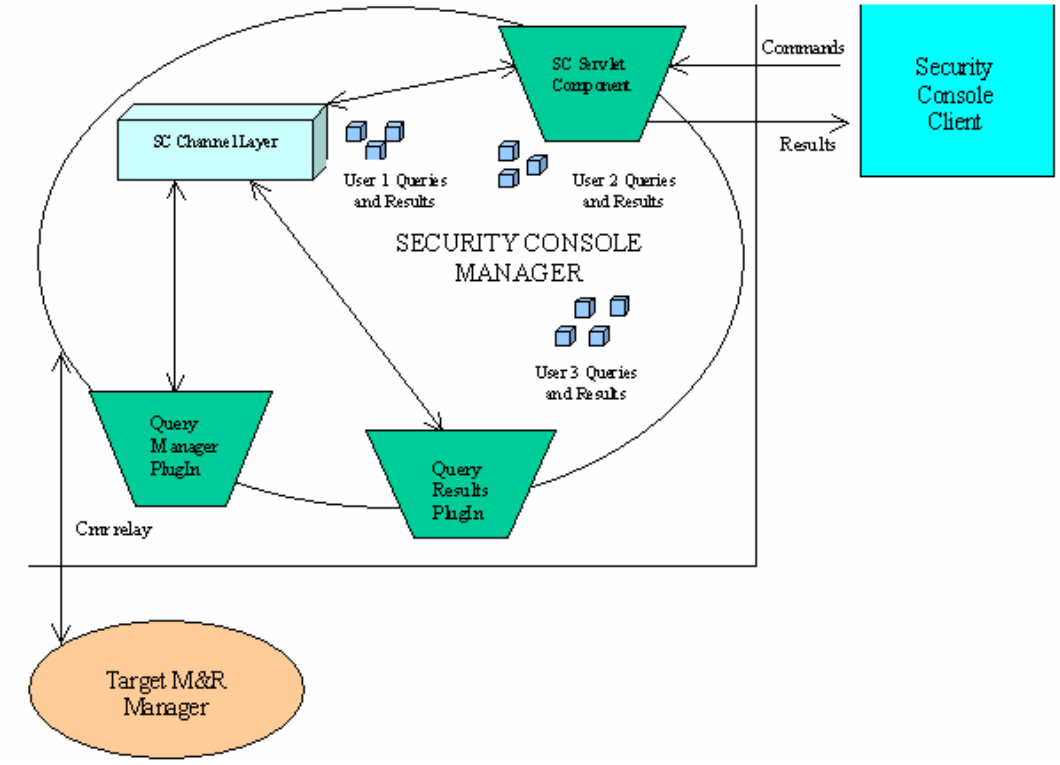


Figure 5. Architecture of Security Console Manager

The SC manager is composed of a servlet component and two plugins: Query Manager Plugin and Query Results Plugin. The users, queries and the map between them are maintained in the blackboard object SCChannelLayer.

3.2.2.1 Security Console Servlet Component

This servlet accepts connections from security console clients. The messages received from the SC client are processed here. Except for the messages “OPEN CONNECTION” and “I AM ALIVE” every message is fed to the SC Channel Layer. Open Connection command is received the first time a SC client contacts the SC manager. This servlet component sends acknowledgements to all the messages received for the SC client. The out stream is mapped to the SC client in the sc channel layer. This stream is later used by the other plugins to send information to the SC client.

3.2.2.2 Query

Mainly two types of queries are supported, Aggregation Queries and Expansion Queries.

The aggregation queries have a jython predicate and an xml increment format to format the results. An aggregation query is sent to a MnR Manager to get the consolidated event at the level of that manager. This result may be further expanded to lower levels by sending an expansion query. The expansion query has the information about the point of expansion. The result of this type of query is a list of consolidated and non-consolidated idmef events. The consolidated events in this list may further be expanded, by sending the appropriate expansion queries.

3.2.2.3 Query Manager Plugin

- This plugin receives commands from the SC client and performs the necessary tasks. The following commands are supported.
- Publish Query
- Publish Expansion
- Delete Query
- Get All
- Get Society Info

Publish Query command has an aggregation query and the target manager as parameters. The QueryManagerPlugin publishes a relay for this query. The query has the predicate and the xml increment format.

Publish Expansion command has the information about the point of expansion. A relay with this query is sent to the appropriate target MnR manager.

Delete Query command has the id of the query to be deleted. As query may have expansion queries below them the deletion process is recursive.

Get All command is sent by the SC client to update itself with all the results of its queries. This command is sent every time the SC client connects to the SC manager. The SC manager gets all the events of all the queries for the SC client form the MnR Manger in the society and sends them to the SC client. The QueryManagerPlugin returns the results for all the queries of the SC client at the root level. The results for all the lower level queries are sent to the SC client by the QueyResultsPlugin.

Get Society Info command is sent by the SC client to get the society information. The SC client is especially interested in

1. The names and locations of the SC managers and

2. The name of all the security enclaves and the representing MnR mangers.

3.2.2.4 Query Results Plugin

Query Results Plugin is responsible to send the results of the queries as and when available to the appropriate SC client. The plugin uses the SC Channel Layer to access the map between the users and the queries. The QueryManagerPlugin sends the results for all the root level queries when a Get All command is received. This plugin is responsible to return the results for all queries below the root level. This plugin user the output streams mapped to the users in the SC Channel Layer.

3.2.2.5 SC Channel Layer

This is object is published by the SC Servlet Component at the startup. SC Channel layer provides abstraction to all the http communication. SC Servlet Component, Query Manager Plugin and Query Results Plugin use this object to send/receive data to/from the SC client.

3.2.2.6 Results

The results are sent to the Security Console Client in XML format. The results are a set of idmef objects. These objects are grouped by QID. Each such group corresponds to a query identified by its QID and other information necessary for the SC client to figure out which part of the query result goes to which position in the result trees in the qui. Besides the results of queries the SC manager also sends the status of commands like delete.

3.2.2.7 Target MnR Manager

The target MnR manager is the SC manager where a query is sent. This SC manager is responsible to consolidate all the events satisfying the query at its level and send the resultant consolidated event to the console manger. If it's an expansion query, the list of the entire consolidated and non-consolidated event right below its level is sent to the console manger. This SC manager sends only the delta updates of the events as and when they are available. When, the SC client sends a Get All commands, the console manger sends a request to this manger for a complete set of events for the queries as the console manger has only the latest delta updates in its blackboard.

3.2.2.8 Event Mining Methodology

The core module of Mining Component is responsible for extracting useful information from the logged results. This information is then utilized to find frequent patterns among the events. The Statistics and frequent associations provide the user with important decision making (like specific attack patterns, controlling the network nodes) and monitoring. The mining is done off-line over the aggregated data collected over a period of time in the database.

3.2.2.8.1 Detailed Description – Mining Algorithm

An episode is a collection of events that occur relatively close to each other in a given partial order. We consider the problem of discovering frequently occurring episodes in a sequence. Once such episodes are known, one can produce rules for describing or predicting the behavior of the sequence.

Our Objective is the following:

- Analyzing event sequences is to find frequent episodes i.e., collections of events occurring frequently together. For example, in the sequence of Figure 6, the episode "E is followed by F" occurs several times, even when the sequence is viewed through a narrow window.
- Obtain rules that describe relationships between events in the given event sequence.
- Use the relationships in the on-line analysis of the incoming alert stream, so as to explain the problems that cause alerts, to suppress redundant alerts, and to predict severe damages.

In the analysis of sequences we are interested in finding all frequent episodes from a class of episodes. To be considered interesting, the events of an episode must occur close enough in time. The user defines how close is close enough by giving the width of the *time-window* within which the episode must occur. We define a window as a slice of an event sequence, and we then consider an event sequence as a sequence of partially overlapping windows. In addition to the width of the window, the user specifies in how many windows an episode has to occur to be considered frequent.

Figure 6. A sequence of events.

Here we consider the following problem.

- Knowledge Discovery task: Given a class of episodes and an input sequence of events, find all serial episodes that occur frequently in the event sequence.
- Association Rules and Estimation: Given all frequent episodes, generate rules and their confidences from the frequencies.

To solve the above we did the following,

- Computed the collection of frequent episodes from a class E of episodes. A level-wise (breadth-first) search in the class of episodes is performed following the sub-episode relation. The search starts from the most general episodes, i.e., episodes with only one event. On each level of iteration first a collection of candidate episodes are calculated, and then their Support (threshold frequencies) is checked from the event sequence.
- Given all frequent episodes Episode rule generation ensues and their confidences can be computed. (Given, the frequency threshold we are to find those Episodes that exceed this threshold, to be considered as frequent).

3.3 Dependency Description

3.3.1 Security Console Client

3.3.1.1 GUI Component

Main Interface in the GUI component shows all other components. It also provides a well-built interface to query IDMEF objects from M&R manager. The user-entered query is sent to query component, which retrieves data from M&R manager. This data can be depicted in different views using the Visualization tools.

3.3.1.2 Query Component

Because all possible sources of IDMEF objects can be referenced by an id (society, urgent alerts, and query results), the query component receives the query sentence and the **id** of the IDMEFs source from the GUI component. The GUI component can communicate with the Query Component only through the query manager module. It then returns to the GUI component manager and the set of Java IDMEF that satisfy the query predicate (query sentence).

If the IDMEFs source's id is the society id, the query manager sends to the society, the query sentence that will be used to select the objects. Then the manager and society return the id of the query result composed by the IDMEF objects that satisfy such query sentence. When new objects are available for such query result, they are added.

If the IDMEFs source's id is other than the society id, the query component creates a query result (and an id for it) that is the result of applying the query sentence over the given query result.

3.3.1.3 Communication Component

While the query component task is to send to the communication component a query sentence and receive a set of Java IDMEF objects, the communication component on the other hand sends to the M&R manager a query command and receives a set of XML IDMEF objects and XML.

The following sequence diagrams illustrate the inter-dependency of the security console components.

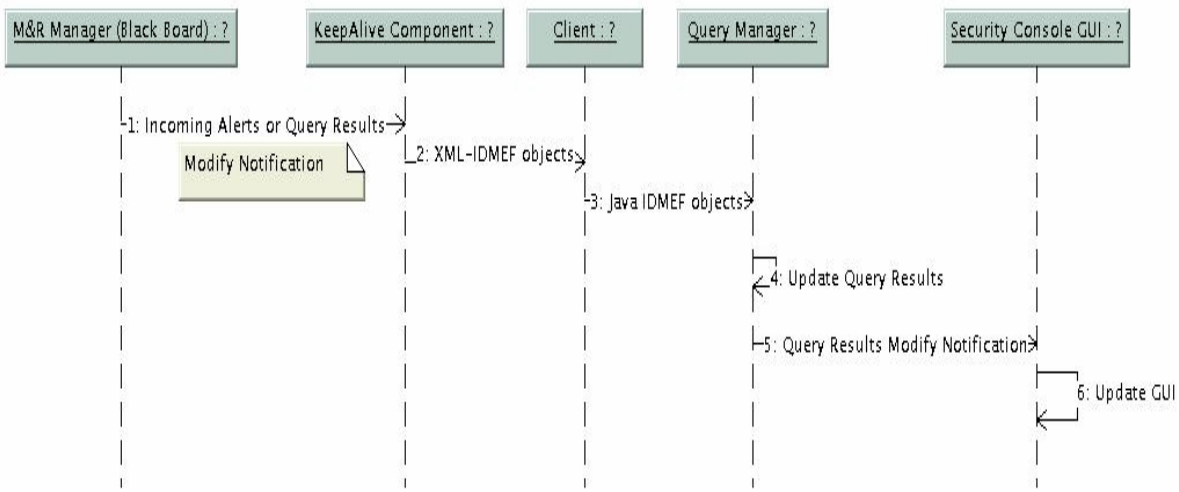


Figure 7: Obtaining results using Keep-Alive Component

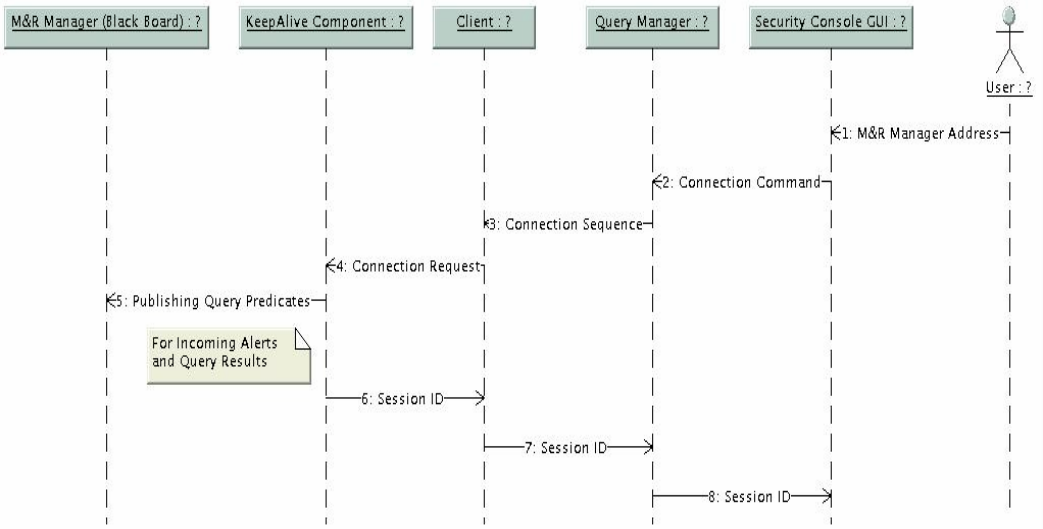


Figure 8: Establishing the connection with the Keep-Alive Component

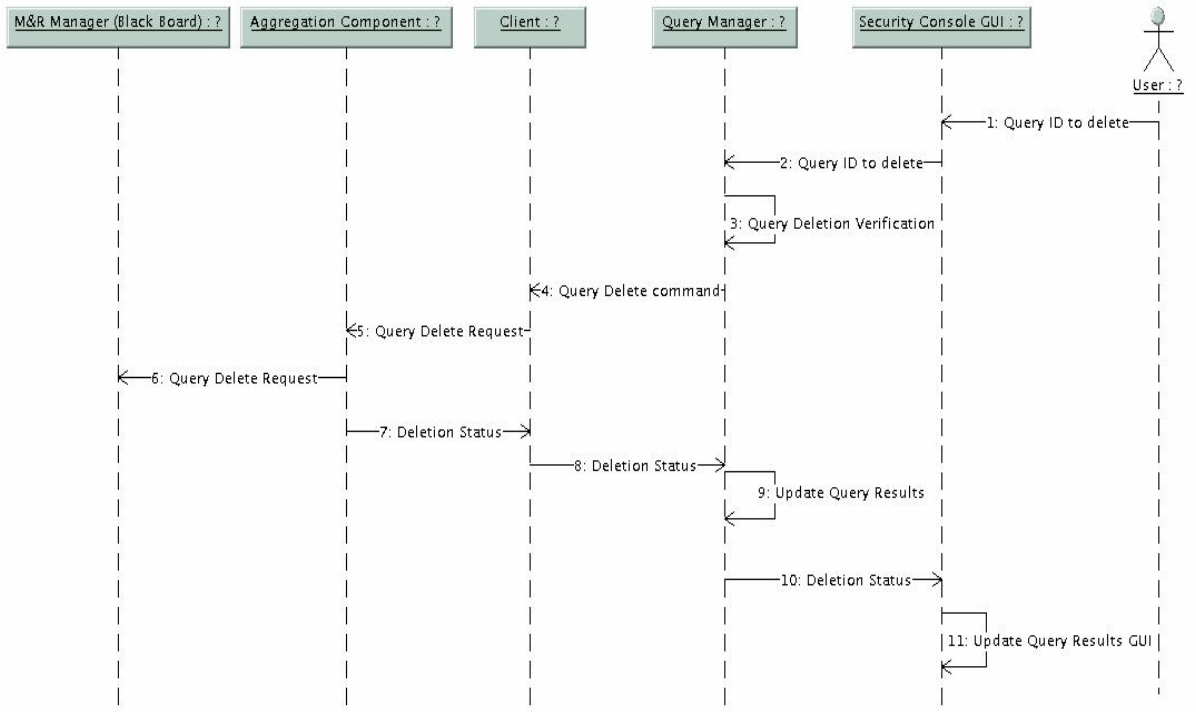


Figure 9: Query Deletion

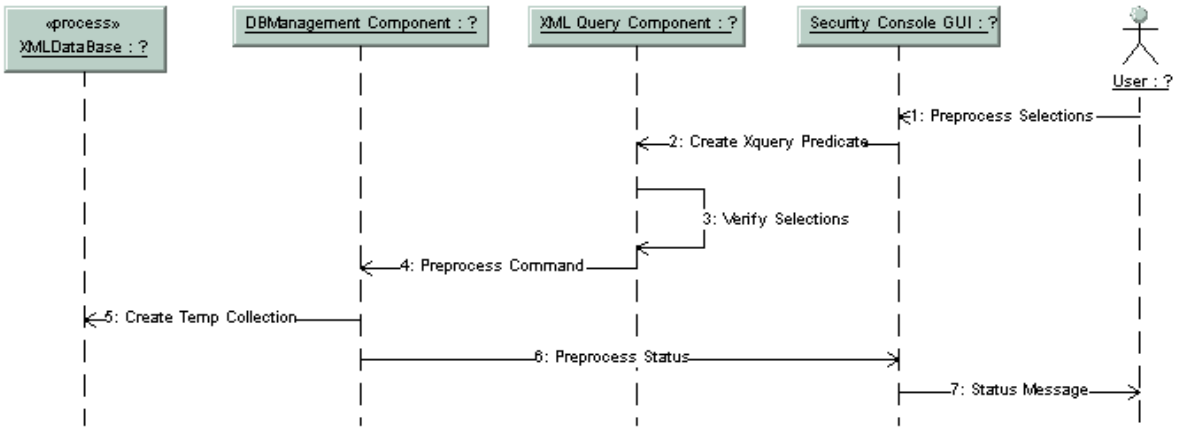


Figure 10: Data Base Preprocess

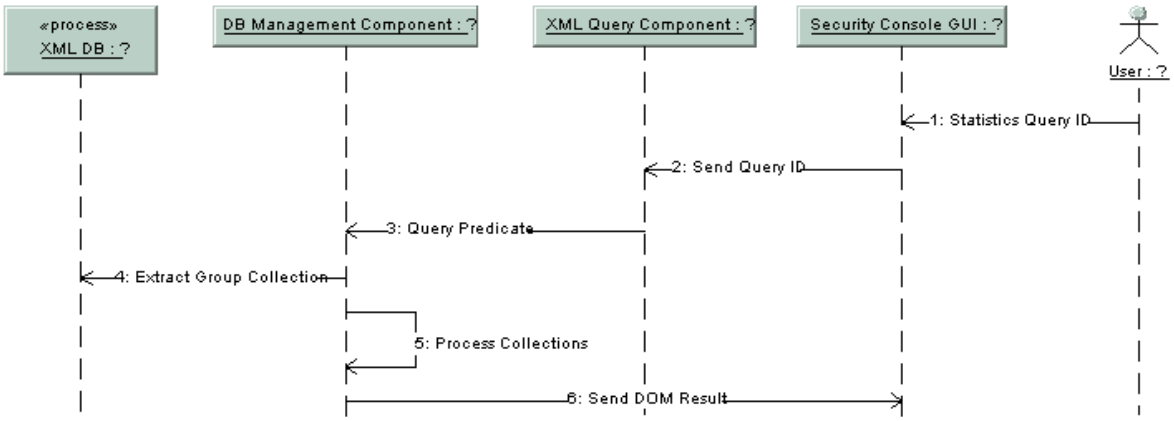


Figure 11: Collect Message Statistics

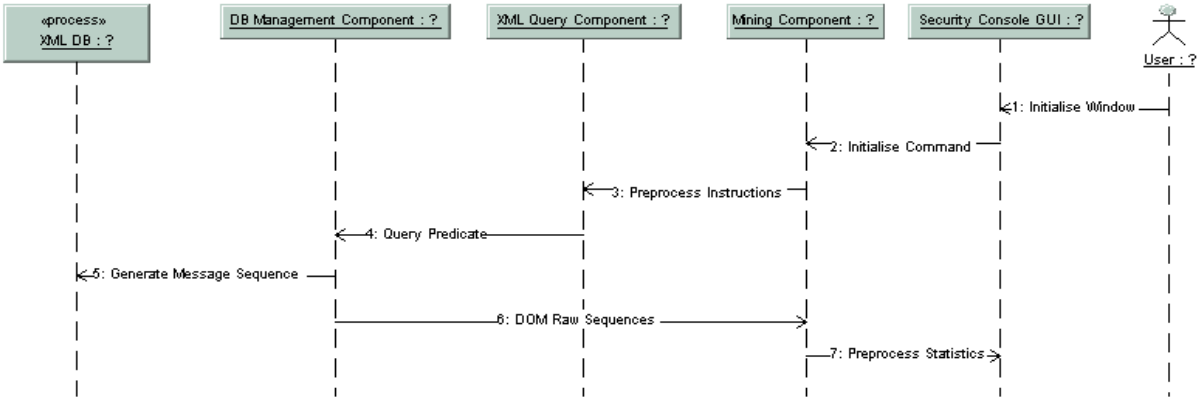


Figure 12: Initialize Mining Sequence

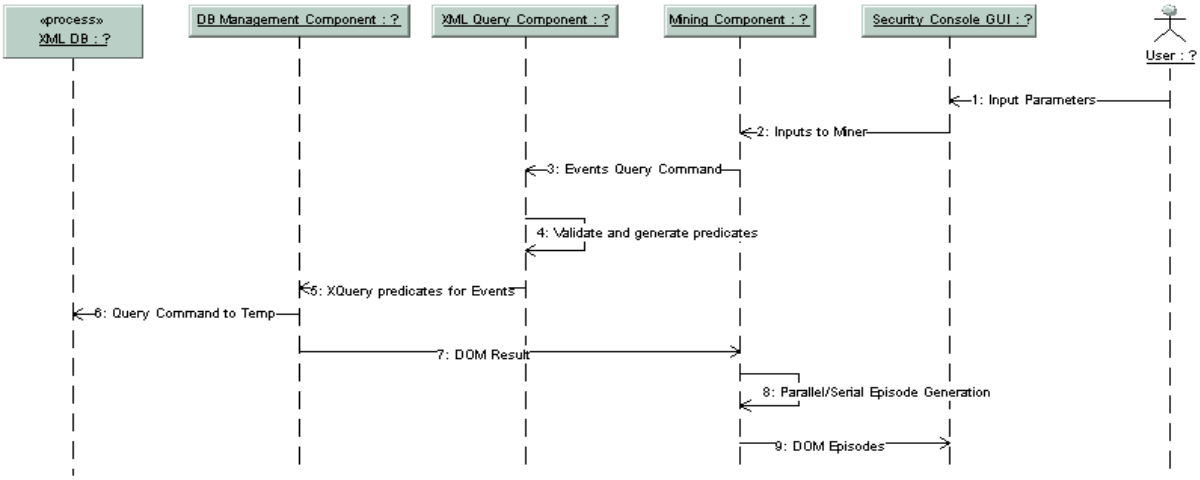


Figure 13: Mining Alert Message

3.3.1.4 Data Base Management Component

This component is associated with an XML Database from 'eXist' for doing off-line Mining. It basically communicates on one end with the GUI component while on the other hand interfaces with the database through the dbDriver. The database Manager is responsible for establishing and maintaining connection to the database for any message exchange. The manager is also responsible for retrieving the high level view of the database organization for the collections', to the GUI component. The component is equipped with the capability to iterate through the XML collections structure in the database and sieve out the messages as appropriate.

The XML Query Component creates complex query structures and this component takes it as the input to process the messages' collection database and returns a Document Object Model, back.

3.3.1.5 XML Query Component

This component uses the XQuery tool for generating complex query structures to communicate with the Database. The results are sent to the GUI component (Statistics, Mining) for displaying the appropriate Visualization. This is achieved in the following ways:

- On getting a specific query command from the GUI component it creates an XQuery predicate and sends it to the DB management component.
- Sends the Result to the GUI component for Visualization.
- Communicates with the mining component for creation of a XQuery predicate and send it to the DB management component for retrieving messages of interest.
- Sends the Result to the Mining Component for further Processing.

All the basic data communication involves the exchange of data units as a javax DOM document that can be iterated and queried repeatedly by various modules as and when desired.

3.3.1.6 Mining Component

This component uses XML Query Component for the initialization step that involves the generation of event sequences. This step is the Pre Processing phase of Mining. The GUI component provides the inputs for mining (serial or parallel). The episode mining results are sent back to GUI component for visualization. All the parameters that it receives from the GUI Component are fed to it by the User. The results include Rules as well. The related events of the episode are connected by some pattern using Association Rules that this component returns for visualization and/or further initialization.

3.4 Interface Description

3.4.1 Security Console Client

3.4.1.1 GUI Component

GUI component takes the lead role, acting as a commencing point where the user enters the desired query. It incorporates 'Helper GUIs' to assist the user in creating, editing and storing the queries. The above functionalities are modularized into 'SnapIn's and can be loaded into the Main Interface by editing the properties file. The GUI also offers different views of the data like TreeView and TextView SnapIns, which are provided by default. A component willing to support a view should implement the 'View' interface.

3.4.1.2 Query Component

The query component receives a string with a predicate that is used to get the IDMEF objects and the aggregation method to be applied by the drill down infrastructure. The first string contains a corresponding Jython function that uses the Java IDMEF library and the Aggregation agent infrastructure. The second one contains the specific parameters for the aggregation task done by the M&R managers.

This component receives all the commands that can be sent to the SC Manager. A list of all the commands can be found in section 3.4.2

3.4.1.3 Communication Module

This component creates a security console message for each command that is sent by the Query component to the SC manager. This message has the following XML format:

```
<SC-Message>

 <SC-Source id="User1" />

 <SC-Target id="SCM4" />

<SPECIFIC COMMAND/>

</SC-Message>
```

The source corresponding with the id of the user that is sending the command. The target is the SC manager used as mediator. This message is part of a URL string connection as shown below:

http://host:8800/$SCmrmanager/manager?THICK_CLIENT=1&CONNECTION=1

The communication component receives from the SC manager a XML document with a SC message with query results, command status or possible errors.

3.4.1.4 Data Base Management Component

This communicates with the database through XQuery predicates. The input to the '*eXist' Database* is a sequence of XML messages. This sequence comes from two sources. One directly from the Query Results plugin. The other from the Log File that is done off-line.

The setup is done in such that each IDEMF Alert message coming from the society is stored in the following path: */db/date/hour/queryId/sensorid/alertId.xml*. The root collection is *db*; *date* is in the form *yyyy-mm-dd* and represents the creation time of the IDEMF message; *hour* represents the creation time of the IDEMF message; *queryId* represents the query identification and finally *alertid.xml* represents the IDEMF alert identification.

The database has a structure of collections similar to a directory structure in UNIX. For example the UNIX directory path /db/2000-01-01/23/SecurityAgent is represented in eXist XML database as three collections db, 2000-01-01 and SecurityAgent. SecurityAgent is a child collection from 23 and 2001-01-01 is also a child collection from db and so on. To access the XML documents inside eXist DB SC uses XQuery.

3.4.1.4.1 Data Base Manager

The database records are manipulated through the database manager that is responsible to make and initialize connection through the database native Driver. Once the components are linked to the database, the manager can process commands over database records. The basic commands are *update Database*, *preprocess*, *delete select records* and *delete temporary collection*.

The Database manager takes as its input XQuery predicate and alters the records according to the command sent. For *inserting* new collections into the database, the manager handles the insertion in the right fashion.

An example of the *preprocess* command as XQuery predicate is in this format:

```
" let $col := collection(\"/db/" + path + "\"" + ")  "  ;

"     for     $a     in     distinct-values($col/ANALYZER-RESULTS/IDMEF-
Message/Alert/@ident) "  ;+
" for $b in $col/ANALYZER-RESULTS/IDMEF-Message/Alert[@ident=$a]  " ;+
" return <Alert ident=\""+ "{string($a)}" + "\">" ; +
"  {  for  $An  in  $b/Analyzer/@analyzerid  return  <Analyzer  ident=\""+
"{($An)}\">" + "</Analyzer> }   " ;+
"  {  for  $Ti  in  $b/DetectTime  return    <DetectTime  time=\""  +
"{string($Ti)}\"" + "timehour=\"" + "{substring(string($Ti),1,13)}\""  +
"timemin=\""  + "{substring(string($Ti),1,16)}\"" +  ">" + </DetectTime>
}   " ;+
" { for $So in $b/Source/Node/Address/address return <Source ident=\""+
"{string($So)}\">" + "</Source> }   " ; +
" { for $Ta in $b/Target/Node/Address/address return <Target ident=\""+
"{string($Ta)}\">" + "</Target> }   " ; +
"  {  for  $Cl  in  $b/Classification/name    return    <Classification>
{string($Cl)} </Classification> }   " ; + " </Alert> " ;
```

Database Table Result:

```
<Result><Date value = “2004-09-29”>

 <Hour value = "24" >

      <Query id = “SCM1/123456789”>

           <Numbers value = “60”/>

      </Query>

 </Hour>
```

```
<Hour value = "24" >

      <Query id = “SCM1/123456789”>

            <Numbers value = “60”/>

      </Query>

</Hour>
```

</Date></Result>

3.4.1.5 XML Query Component

This component provides the message communication from GUI component and the Mining Component to the Database. It interacts with the Data base manager that ultimately returns the *XML string* containing the result that is modified *as DOM* and returned to the querying components accordingly.
For the statistics module, this component after receiving a specific request query from the GUI component, creates an XQuery predicate and sends it to the database manager for retrieving the XML result as DOM. The queries are Sequence Message, Source Activity, Target Activity and Analyzer Activity.
An example of XQuery predicate for *Source Activity* is in this format:

```
" let $a:= collection(\"/db/" + "Temp" + "\"" + ")/Result  " ; +
"  for  $b  in  distinct-values($a/Alert/Source/@ident)  ,  " ;  +  "  $c  in
count($a/Alert/Source[@ident=$b]) order by $c descending  " ; +
" return <Source ident=\"" + "{string($b)}\""  + " total=\"" + "{($c)}"
+ "\">" ; +
" { let $d := $a/Alert/Source[@ident=$b]/parent::* " ; + "    let $e :=
distinct-values($d/DetectTime/@"+ attr   + ")" ; + "    for  $f  in  $e
order  by  $f    return  <DetectTime>  {$f}      "  ;  +  "  {  for  $g  in
count($d/DetectTime[@"+ attr + "=$f]) return <Count> {$g} </Count> }";+
" </DetectTime> }  </Source> " ;
```

An example of XQuery predicate for *Analyzer Activity* is in this format:

```
" let $a := collection(\"/db/" + "Temp" + "\"" + ")/Result  " ;+
" for $b in distinct-values($a/Alert/Analyzer/@ident) ,    " ; +
" $c in count($a/Alert/Analyzer[@ident=$b]) order by $c descending ";+
"  return  <Analyzer  ident=\""  +  "{string($b)}\""    +  "  total=\""  +
"{($c)}" + "\">" ; +
" { let $d := $a/Alert/Analyzer[@ident=$b]/parent::* " ; +
"  let $e := distinct-values($d/DetectTime/@timehour) " ; +
"   for $f in $e  order by $f  return <DetectTime> {$f}   " ; +
" {  for  $g  in  count($d/DetectTime[@timehour=$f])  return  <Count>  {$g}
</Count> } " ; + "  </DetectTime> }  </Analyzer> " ;
```

Analyzer Activity Result Format:

```
<Result>
      <Group>
            <Analyzer ident = “SecurityAgent-4-2/Test” total=”10”>
```

<DetectTime>2004-09-06T01 <Count>9</Count> </DetectTime>
</Analyzer>
</Group>
</Result>

Source Activity Result Format:
<Result>
<Group>
<Source ident = “192.10.3.10 total = “2”>
<DetectTime>2004-09-07T01<Count> 3 </Count></DetectTime>
</Source>
</Group>
</Result>

Time Sequence Result Format:
<Result>
<Sequence count = “120”>
<DetectTime count=”4” time = “2004-09-04T00:01”/>
</Sequence>
</Result>

3.4.1.6 Mining Component

The initialization step is a message to XML Query component to generate a sequence of events for the selected messages that the user is interested to perform mining upon. The XML Query component builds the predicate in response to the generation of event sequence whose format is the following:

```
" let $col :=  document(\"/db/Temp/"+ _doc + "\"" + ")/Result " ;+
"  let  $a  :=  distinct-values($col/Alert/@ident)  "  ;  +  "  for  $b  in
$col/Alert[@ident=$a] ,  " ;+ " $c in $b/@ident , " ; +
"$d in $b/DetectTime/@time, " ; + " $e in $b/Source/@ident , " ; +
"$f in $b/Target/@ident , " ; + " $g in $b/Classification " ; +
" return <Alert  ident=\"" + "{string($c)}" + "\">" ; +
" {<DetectTime> {string($d)} </DetectTime>} " ; +
"   {<Source>   {string($e)}   </Source>}   ";   +"   {<Target>{string($f)}
</Target>} " ;+ "{<Classification> {string($g)} </Classification>}";+
"</Alert>";
```

The Mining component performs the mining (Serial or Parallel) as per the user information received from the GUI component. The immediate results of this component are frequent episodes. These episodes are communicated back to the GUI component for visualization.

The episodes can be further processed for finding the Association Rules among the candidate episodes. This is the ultimate result of the Mining component which is sent to the GUI component for visualization.

The Episodes format:

```
<Result><EpisodeLength>

 <Group>
```

```
            <Episode Support = “0.432765897”/ >
            <Event>
                  <Attribute1>192.100.55.1</Attribute1>
                  <Attribute2>255.255.200.100</Attribute2>
                  <Attribute3>org.cougaar.Exception1</Attribute3>
            </Event>
            </Episode>
 </Group>
```

</EpisodeLength></Result>

The Rules format:

```
<RULES>
 <RULE>
  <LHS>
   <EVENT Attr1=“192.100.55.1”Attr2=”255.255.200.100”Attr3=“org.cougaar.Exception1”/>
  </LHS>
  <RHS>
  <EVENT Attr1=”192.100.54.21” Attr2= “255.255.100.11” Attr3=“org.cougaar.attack2” />
</RHS>
        <CRITERE name=”Support” value=”10” />
        <CRITERE name =”Confidence” value=”65” />
</RULE>
  </RULES>
```

3.4.2 Security Console Manager

3.4.2.1 SC Servlet Component

The purpose of this servlet component is to open connections and send/receive messages to/from clients. The format of the open, close messages are given below:

Open Connection Message:

```
<SC-Message>
 <SC-Source id="User1" />
 <SC-Target id="SCM4" />
<OPEN/>
</SC-Message>
```

Close Connection Message:

```
<SC-Message>
```

```
 <SC-Source id="User1" />
 <SC-Target id="SCM4" />
<CLOSE/>
</SC-Message>
```

Acknowledge Message:

```
<SC-Message>
 <SC-Source id="User1" />
 <SC-Target id="SCM4" />
<ACK/>
</SC-Message>
```

3.4.2.2 Query Manager Plugin

The QueryManagerPlugin receives and processes the commands: Get-All, Publish Query, Delete Query and Expand Query

These commands are java objects received originally in xml, which are converted to java objects. Shown below are the xml representations of these objects:

Get All:

```
<SC-Message>
 <SC-Source id="User1" />
 <SC-Target id="SCM4" />
<GET-ALL/>
</SC-Message>
```

Publish Query:

```
<SC-Message>
 <SC-Source id="User1" />
 <SC-Target id="SCM4" />
<PUBLISH target_manager="SecurityManager-1">
<query     type="Persistent"     update_method="Push"     pull_rate="-1"
name="name"><unary_predicate language="JPython">from java.util import *
from edu.jhuapl.idmef import *
from org.cougaar.core.security.monitoring.blackboard import Event
from org.cougaar.core.security.monitoring.idmef import RegistrationAlert
from java.lang import Object
from java.lang import String
from org.cougaar.lib.aggagent.query import ResultSetDataAtom
from java.lang import String
def getDocument (x):
  if not (isinstance(x, Event)):
    return 0
```

```
  else:
    try:
      idmefMessage = x.getEvent()
      if (idmefMessage.getClass()!=Alert): return 0
      value = 1

      return value
    except Exception,e:
      print &amp;quot;[Jython exception]:&amp;quot;,e
      return 0
def instantiate ():
  return  getDocument</unary_predicate><xml_encoder  language="JPython"
type="Increment">from       org.cougaar.lib.aggagent.query       import
ResultSetDataAtom
from org.cougaar.core.security.monitoring.blackboard import Event
from org.cougaar.core.util import UID
def encode (out, sacc):
  out.setReplacement(1)
  if (sacc.getAddedCollection()):
    i = sacc.getAddedCollection().iterator()
    while (i.hasNext()):
      da = ResultSetDataAtom()
      event =  i.next()
      from org.cougaar.core.util import UID
      uid = event.getUID()
      da.addIdentifier(&amp;quot;owner&amp;quot;, uid.getOwner())
      from java.lang import String
      da.addIdentifier(&amp;quot;id&amp;quot;, String.valueOf(uid.getId()))
      da.addValue(&amp;quot;source&amp;quot;, event.getSource().toAddress())
      da.addValue(&amp;quot;event&amp;quot;, event.getEvent().toString())
      out.getAddedList().add(da)
def instantiate ():
  return encode
</xml_encoder></query>
<AggregationType name="MnRAggRateCalculator">
<parameters>
<parameter key="timewindow" value="300"/>
</parameters>
</AggregationType>

</PUBLISH>
```

```
</SC-Message>
```

Delete Query:

```
<SC-Message>
 <SC-Source id="User1" />
 <SC-Target id="SCM4" />
<DELETE qid="SCM4/1060973707632"/>

</SC-Message>
```

Expand Query:

```
<SC-Message>
 <SC-Source id="User1" />
 <SC-Target id="SCM4" />
<PUBLISH_DETAILS                    originator_id="SCM4/1060973707631"
parent_id="SCM4/1060973707631" target_manager="SecurityManager-4">
</PUBLISH_DETAILS>

</SC-Message>
```

3.4.2.3 Query Results Plugin

This plugin returns results of the queries to the SC client. Along with the results additional information related to the results that will help the SC client insert the results in the tree is sent.

Query Result of an Aggregation Query (at root):

```
<SC-Message>
 <SC-Source id="SCM4" />
 <SC-Target id="User1" />
<QUERY-RESULTS user="User1">
<QUERY-RESULT      id="SCM4/1060973707631"      mnr="SecurityManager-4"
parent="null" root="null">
<ANALYZER-RESULTS analyzer="SecurityManager-4" type="MnR">
<IDMEF-Message version="1.0">
    <Alert ident="SecurityManager-4/1060973702131">
        <Analyzer/>
        <CreateTime           ntpstamp="0xc2e7abda.0x76200000">2003-08-
15T18:58:34Z</CreateTime>
        <DetectTime           ntpstamp="0xc2e7abda.0x76100000">2003-08-
15T18:58:34Z</DetectTime>
        <AdditionalData                    meaning="CONSOLIDATED_EVENTS"
type="boolean">true</AdditionalData>
        <AdditionalData                        meaning="ORIGINATORS_UID"
type="string">SCM4/1060973707631</AdditionalData>
```

```
        <AdditionalData                                meaning="PARENT_UID"
type="string">SCM4/1060973707631</AdditionalData>
        <AdditionalData                    meaning="TOTAL_CURRENT_EVENTS"
type="integer">0</AdditionalData>
        <AdditionalData                             meaning="TOTAL_EVENTS"
type="integer">0</AdditionalData>
        <AdditionalData meaning="RATE" type="real">0.0</AdditionalData>
        <AdditionalData meaning="AGENT_INFO" type="xml">
            <Cougaar:Agent

Cougaar:class="org.cougaar.core.security.monitoring.idmef.Agent"
Cougaar:name="SecurityManager-4">
                <Address category="url">
                    <address>SecurityManager-4</address>
                </Address>
                <Cougaar:ref-ident/>
            </Cougaar:Agent>
        </AdditionalData>
    </Alert>
</IDMEF-Message>
</ANALYZER-RESULTS>
</QUERY-RESULT>
</QUERY-RESULTS>


</SC-Message>
```

Query Result of an Expansion (at the leaf level):

```
<SC-Message>
 <SC-Source id="SCM4" />
 <SC-Target id="User1" />
<QUERY-RESULTS user="User1">
<QUERY-RESULT      id="SCM4/1060974181237"      mnr="SecurityManager-4"
parent="SCM4/1060974181236" root="SCM4/1060974181236">
<ANALYZER-RESULTS analyzer="SecurityAgent-4-2" type="Sensor">
<IDMEF-Message version="1.0">
    <Alert ident="SecurityAgent-4-2/1060974183430">
--------------- “IDMEF alert goes here” ----------------------------
      ------------------------------------------------------------
--------------- “End of the alert” ---------------------------------
    </Alert>
</IDMEF-Message>
```

```
</ANALYZER-RESULTS>
</QUERY-RESULT>
</QUERY-RESULTS>

</SC-Message>
```

4 Detailed Design Description

4.1 General Description

Security console (SC) is used to generate queries and display results in the form of IDMEF objects. Different sensors in the society generate IDMEF objects when security violation occurs or at regular intervals of time (consolidate events). These IDMEF objects can be displayed in different view modes. Accordingly, Security console provides interfaces for specifying the user and the Security Console (SC) managers, to generate multiple aggregated queries and for retrieving IDMEF objects.

The time-stamped messages are collected in a repository to mine and extract features of importance. The messages are analyzed to reveal important relationships and help the user make crucial decisions regarding the activity patterns in the society. This information can then be statistically analyzed with components that display “mined” data in statistical formats.

4.2 SC Components

4.2.1 Main Interface

Main Interface starts when application is started. It has a properties file in which all the SnapIns are listed. SnapIn is a GUI component, which has specific functionality. For a component to be shown in Main Interface as SnapIn, it must implement ‘SnapIn’ interface and should be listed in properties file. Similar functionality SnapIns are grouped together into ‘Racks.’ For example Query related SnapIns are grouped in Query Rack.

4.2.2 Setup SnapIn

This is the first task the user needs to do before using the Security Console. Setup SnapIn provides a GUI to set users and SC managers. It also provides option to indicate the host machine on which the SC manager is loaded which requires the complete URL with the port to be specified.

Figure 9 as shown below allows the user to specify the managers that are running in the host machine or on a remote machine. Once the user clicks the *Add* button, it prompts the user to enter the Console Manager (CM) name and the http address. The http address can be either the IP address of the local host machine or the remote machine followed by the port number. The accepted format for the http address is: <*IP address*> : <*port number*>. If the SC manager is running on the same machine where the console is running, then the IP address can be “localhost”. On clicking the *OK* button, the CM name and the http address are displayed on the Setup SnapIn along with the *Connection Status*, which is initially “No Connected” as shown in Figure 11.

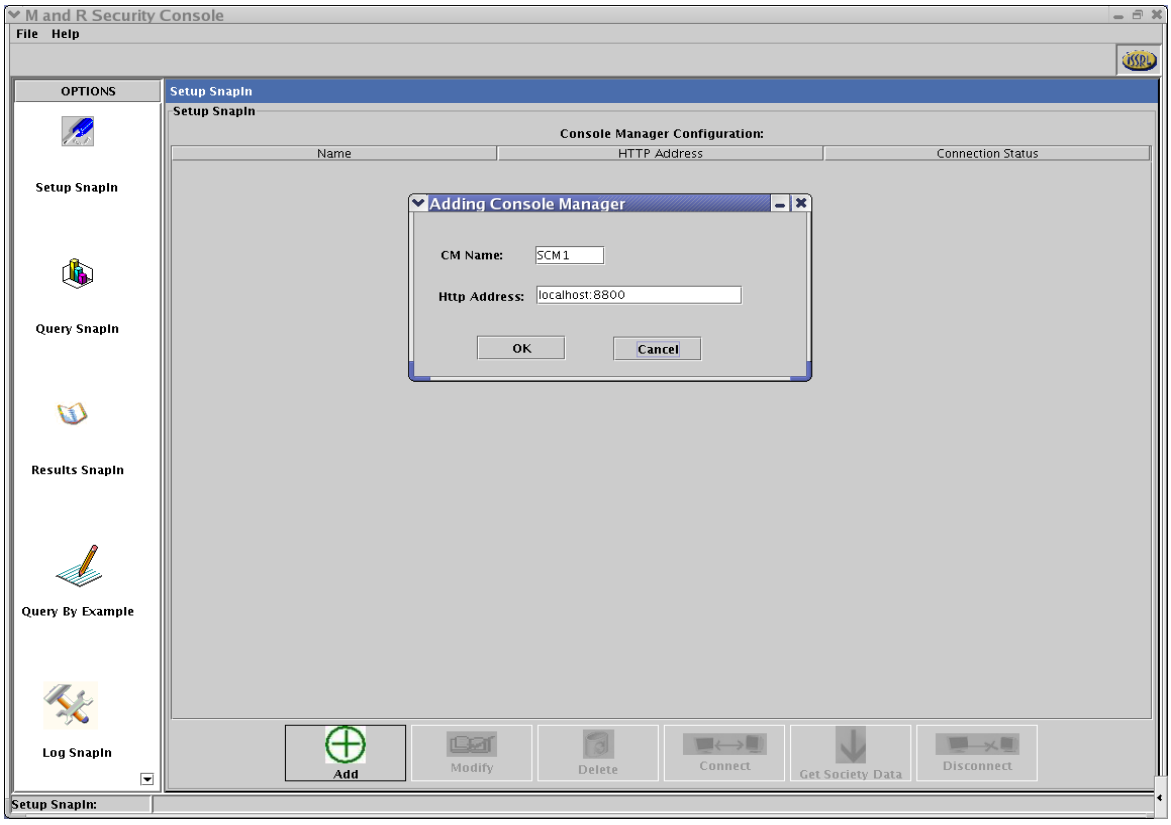


Figure 14: Setup SnapIn adding a Security Console Manager

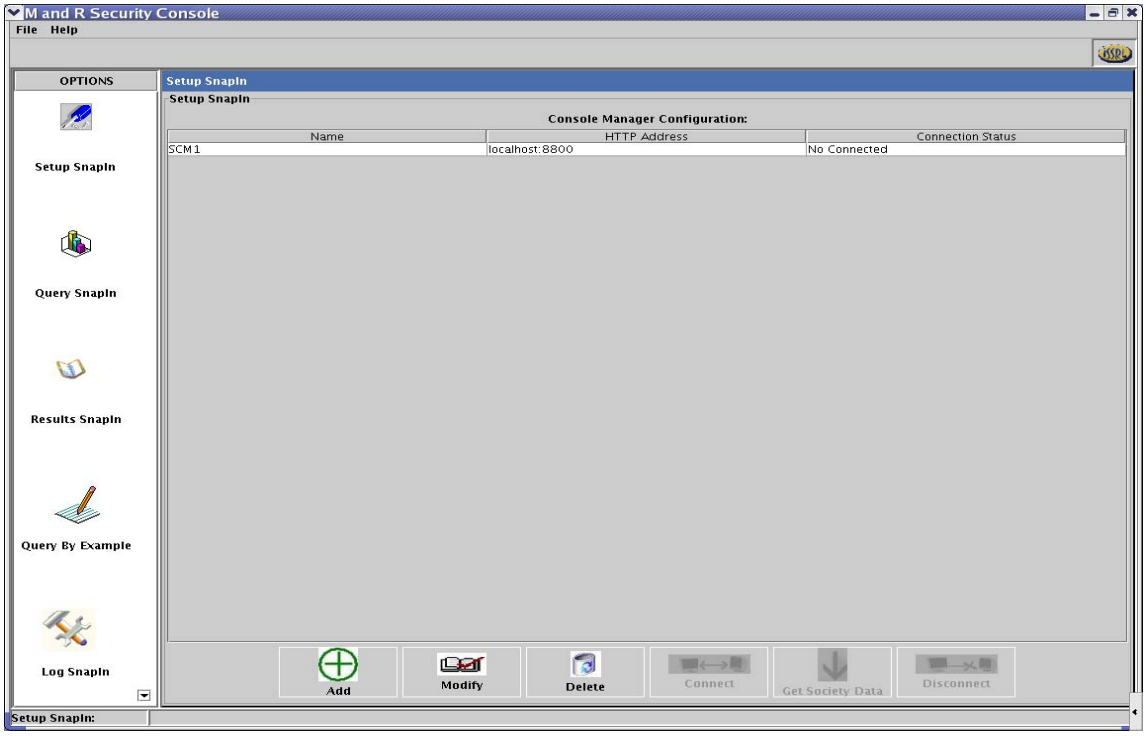


Figure 15: Setup SnapIn without a connection to the Console Manager

In order to enable the connection between the security console and the SC manager the *Connect* button is clicked. If the connection is established, the *Connection Status* changes to "Connected" as shown in Figure 13 (highlighted). Otherwise, the *Connection Status* shows "Establishing Connection", which is also shown in Figure 10. To disable the connection between the security console and the SC manager, the *Disconnect* button should be clicked. After the connection is established, to get the information from the society, the "Get Society Data" button has to be clicked which is shown in Figure 13.

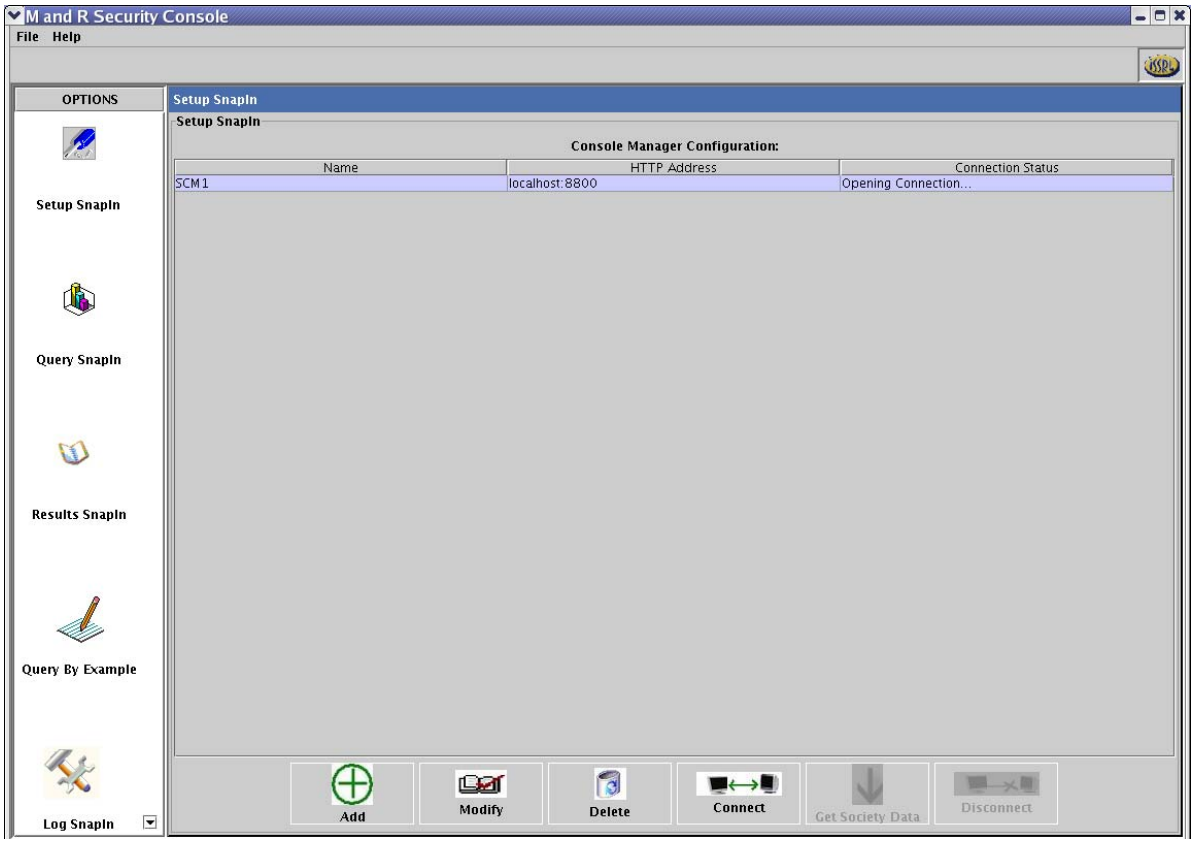


Figure 16: Setup SnapIn opening connection status to the SC Manager

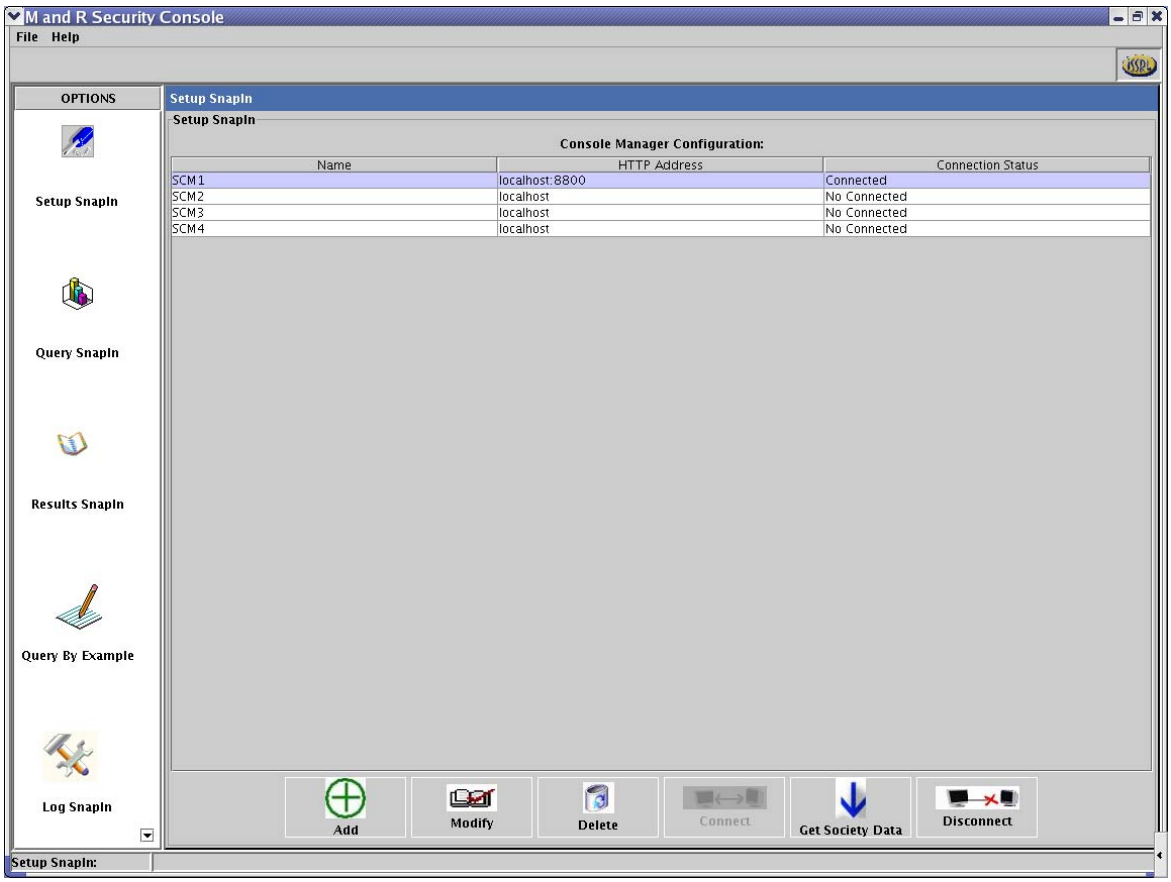


Figure 17: Getting the society data using the Get society Data button.

The user can also modify or delete the manager information using the *Modify* and *Delete* buttons. When the modify button is clicked it presents a dialog box where the new http address can be entered as shown in Figure 14. An important point to notice here is that the user will not be able to modify the CM Name and is only allowed to alter the http address. In order to modify the CM Name, the user has to delete the existing SC manager and then add the new manager. Before deleting, the connection between the SC manager and the security console has to be disconnected using the *Disconnect* button. This notifies the SC manager that the security console is not communicating with it anymore.

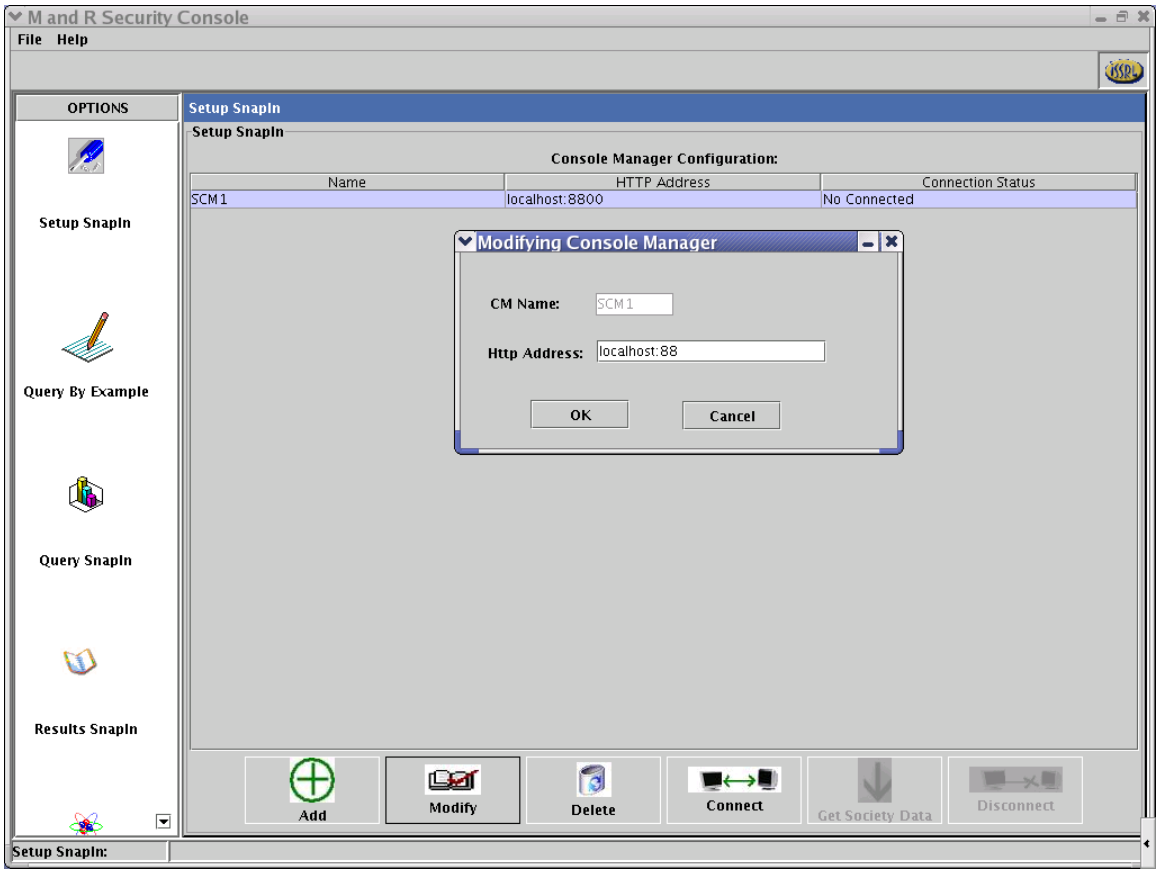


Figure 18: Setup SnapIn modifying the added SC manager

4.2.3 Query SnapIn

The communication between the Security Console Manager (SCM) and the Security Console Client (SCC) is shown in Figure 15. The SCC sends queries in the form of commands to the SCM which inturn fetches the Query Results from the M&R Manager. These results are then returned to the SCC. Thus, the CM acts as a mediator between the SCC and the M&R Manager. In Figure 15 every node has a Security Manager, Console Manager and two Security Agents. The Console Manager of every node communicates with the SCC. It shows four nodes, where Node1 is expanded to Node2 and Node3 and then Node2 is expanded to Node4.

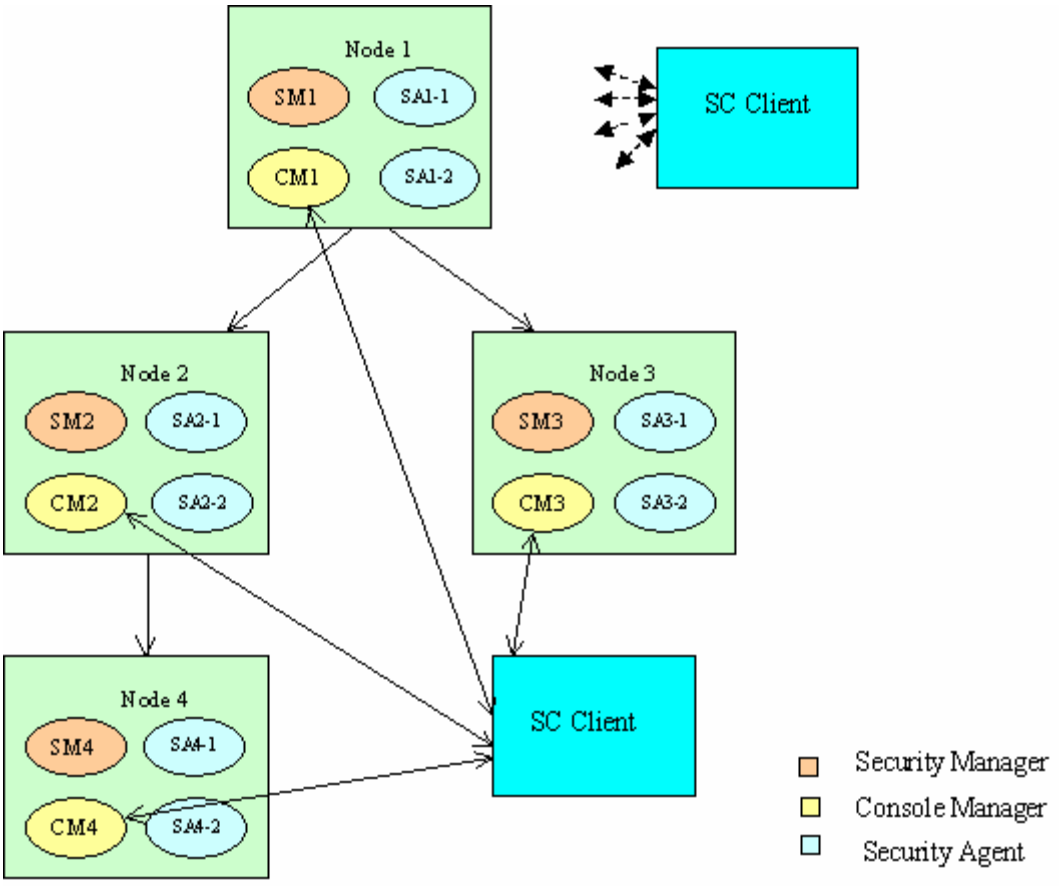


Figure 19: Communication between Security Console Managers and Security Console Clients

4.2.3.1 Query SnapIn Components

Query SnapIn interface consists of two regions(top and bottom) as shown in Figure 16. The top region has three tabbed interfaces,

- PREDICATE
- STORED QUERIES
- INCREMENT FORMAT

The bottom region is the Options which specify the manner in which the query can be aggregated and sent between the SC client and possibly many SC managers. This region is composed of *Aggregation Type* and *Target Console Manager* which are discussed in the following sections.

4.2.3.1.1 Options

Aggregation Type:

It lists the different aggregation types in a drop down list. The aggregation type allows the user to aggregate the queries provided in the PREDICATE interface. When one of the types is selected its corresponding arguments like parameter and value are displayed.. A snapshot of this is shown in Figure 16. Here, the value represents the frequency with which the queries are aggregated in the SC manager.

Target Console Manager and Target M&R Manager:

All the console managers added in the Setup SnapIn by the user are listed (in the list box provided) in the *Target Console Manager*. The user should enter the *Target M&R Manager* that the *Target Console Manager* will be communicating with. A text field is provided for the user to enter the M&R manager name beneath the console manager list box. The user can send the query by choosing Target Console Manager and Target M&R Manager and then by clicking the Send Query button.

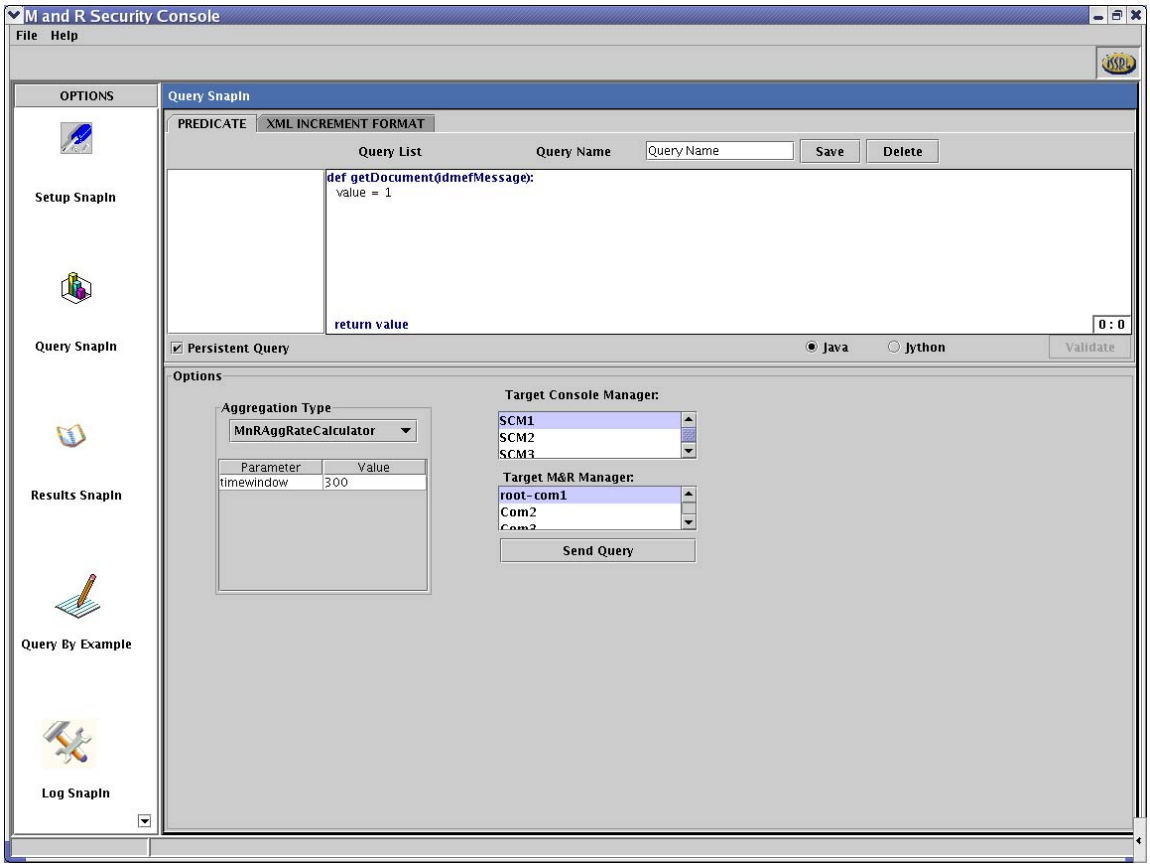


Figure 20: Query SnapIn Predicate to add and edit a query

4.2.3.1.2 PREDICATE Interface

PREDICATE enables the user to enter a query in the query text field. An example (Example#1) is given below specifying the format of the query. Every query is named for identification. Persistence can be applied by checking the *Persistent Query* checkbox. Persistent queries are those, which continuously look for IDMEF objects until they are available and removed. Line number and column number are displayed in the lower right corner of query text area. First line and last line of query are already displayed. They cannot be edited. As indentation is important for python, one space indentation is already done for the query text area. It is also possible to save the queries so that they can be later applied to several different societies or to different SC managers. This is accomplished by clicking on the *Save* button, which opens a dialog box indicating that the query is saved (with the given name) in memory, is displayed. This is shown in Figure 16.

Example#1:

```
def getDocument(idmefMessage):
value = 1
#idmefMessage.getAnalyzer().getAnalyzerid()<"1111"
value = value and (idmefMessage.getAnalyzer().getAnalyzerid()<"1111")
#idmefMessage.getAnalyzer().getProcess().getName()<"someProc"
value= value and  idmefMessage.getAnalyzer().getProcess().getName()<"someProc")
return value
```

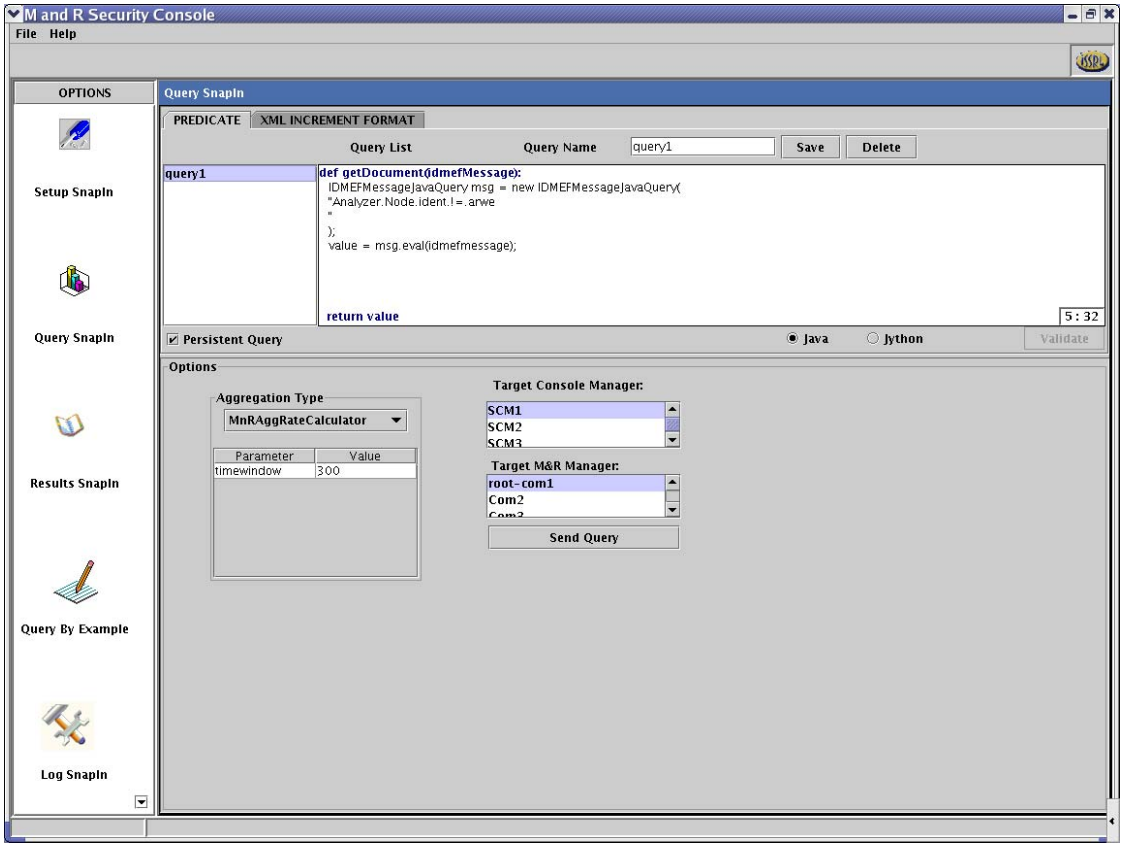


Figure 21: Predicate interface: A query saved to "query1".

Validating a Query

Query validation looks for syntax errors. Any error in query is displayed along with the line number at which the error occurred as shown in Figure 18. The line with the error is highlighted in red color. As python is an interpreter it shows only one error at a time. So it is advised to validate the query till there are no errors.

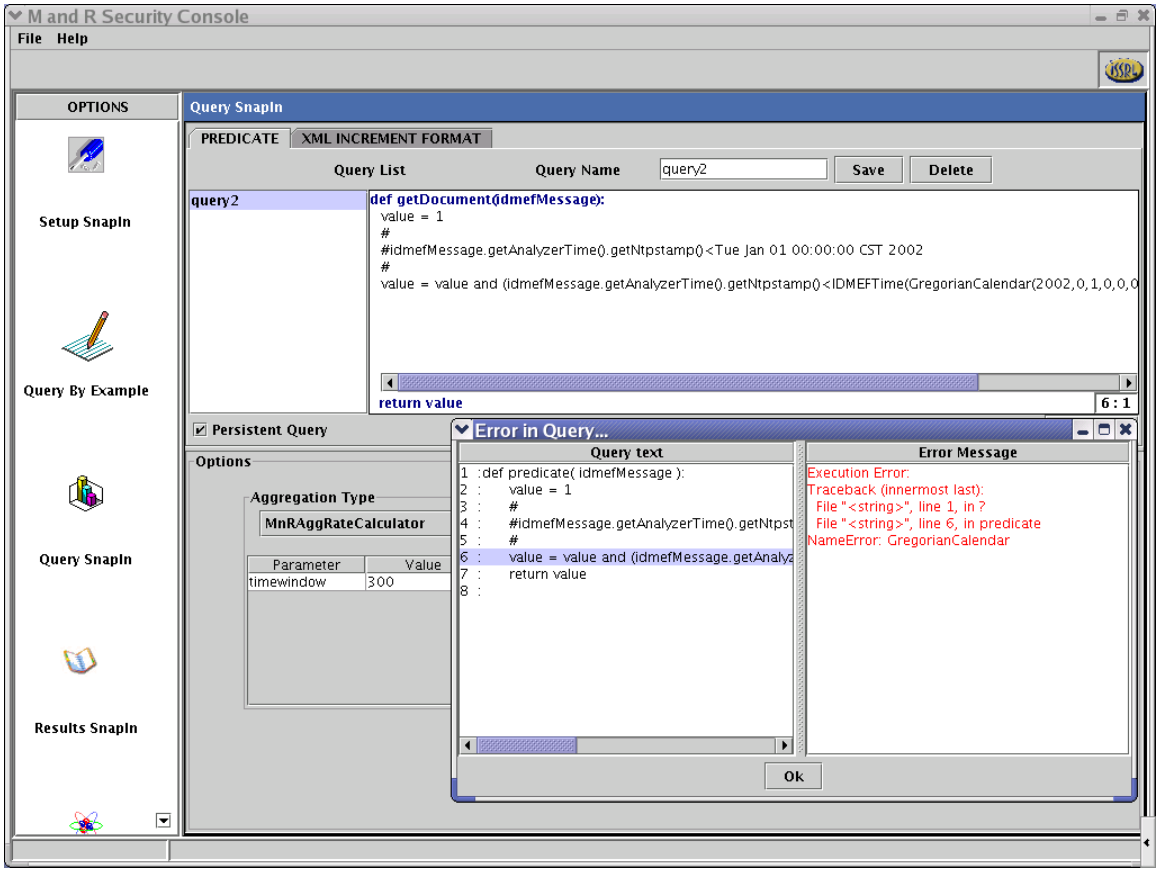


Figure 22: Predicate interface with query validation.

Query saving and retrieving

To save the list of queries to a file, *Save To File* button should be pressed. Already saved queries file can be opened using the *Open* button. If a particular query has to be deleted, the query name has to be chosen first and then the *Delete* button should be used to delete it from the saved queries list.

4.2.3.1.3 INCREMENT Format Interface

Figure 19 shows the snapshot of INCREMENT FORMAT interface. Increment Format is the interface implemented by Objects that convert local subscription data for transportation to other agents (aggregation agents, generally). Information obtained through the SubscriptionAccess interface can be translated in an arbitrary manner into an UpdateDelta, which is then shipped off as an XML document. Refer to Java documentation available at:

Javadoc, org.cougaar.lib.aggagent.session Interface IncrementFormat

The user at Security console client enters just a predicate. The SC client makes an aggregation query with this predicate and an xml increment format and then sends to the console manager. Although, there is no need to change the xml increment format often, the gui allows the user to specify one. This is useful especially when there are changes in the drill down mechanism.

The changes made to the Increment Format can be set using the *Set* button. The *Default* button is clicked to display default increment format.

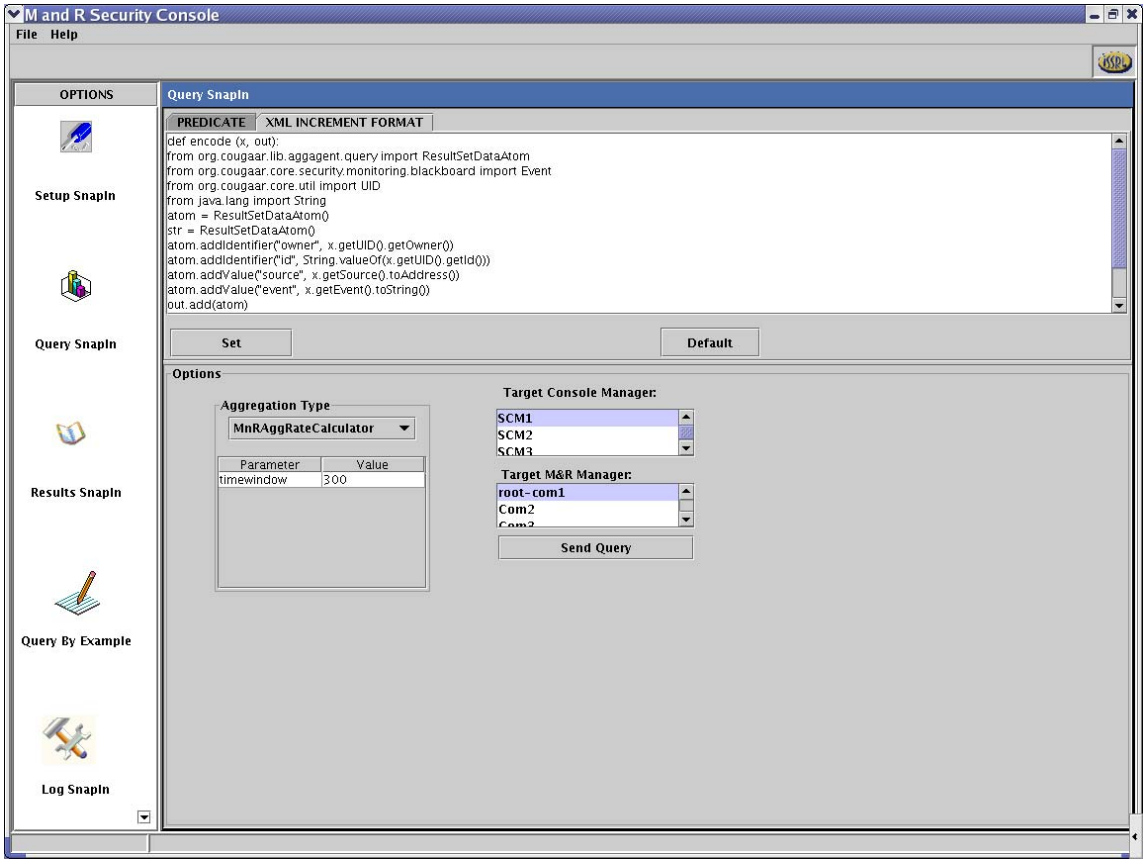


Figure 23: INCREMENT FORMAT

4.2.4 Results SnapIn

After a query is successfully executed the results for that query are displayed in the Results SnapIn interface. A snapshot of this interface is shown in Figure 20. Results SnapIn is composed of three regions. The left most region displays the results in tree format. The top right most region displays a tabular column of attributes for the particular result. The bottom right portion enables the user to view the results in different formats.

Results can be classified into Aggregated results and Non-Aggregated results. The Aggregated results on further expansion yield either Aggregated or Non-Aggregated results. The aggregated results are represented as security managers and the non-aggregated results are represented as security agents in the interface. The results are displayed in three levels, high, middle and low level in a tree format. In order to differentiate between these two types (Aggregated and Non-Aggregated) of results, they are colored differently. The Aggregated results are colored blue and the Non-Aggregated results are colored red. The interface is updated as and when new results are available.

Every high level result represents the aggregated results of the query sent by the user in the Query SnapIn. An example is shown in Figure 20 displaying the results of three queries along with an Id (Id is internal to the Seccon) of the query displayed next to the SC manager name. These queries sent by the user are colored grey and since it is not yet expanded the icon next to the SC manager is displayed as a file icon. In case if it is expanded, the icon is displayed as a folder. Since it is an Aggregated result, the tabular format (to the right) displays the *Meaning*, *Type* and their corresponding *Value* for that aggregated object. Meaning has a set of values like ORIGINATORS_UID, PARENT_UID, etc as shown in Figure 21. Type is a data type, which depends on the value of the Meaning. The lower right most portion displays the consolidated IDMEF Messages using Tree View.

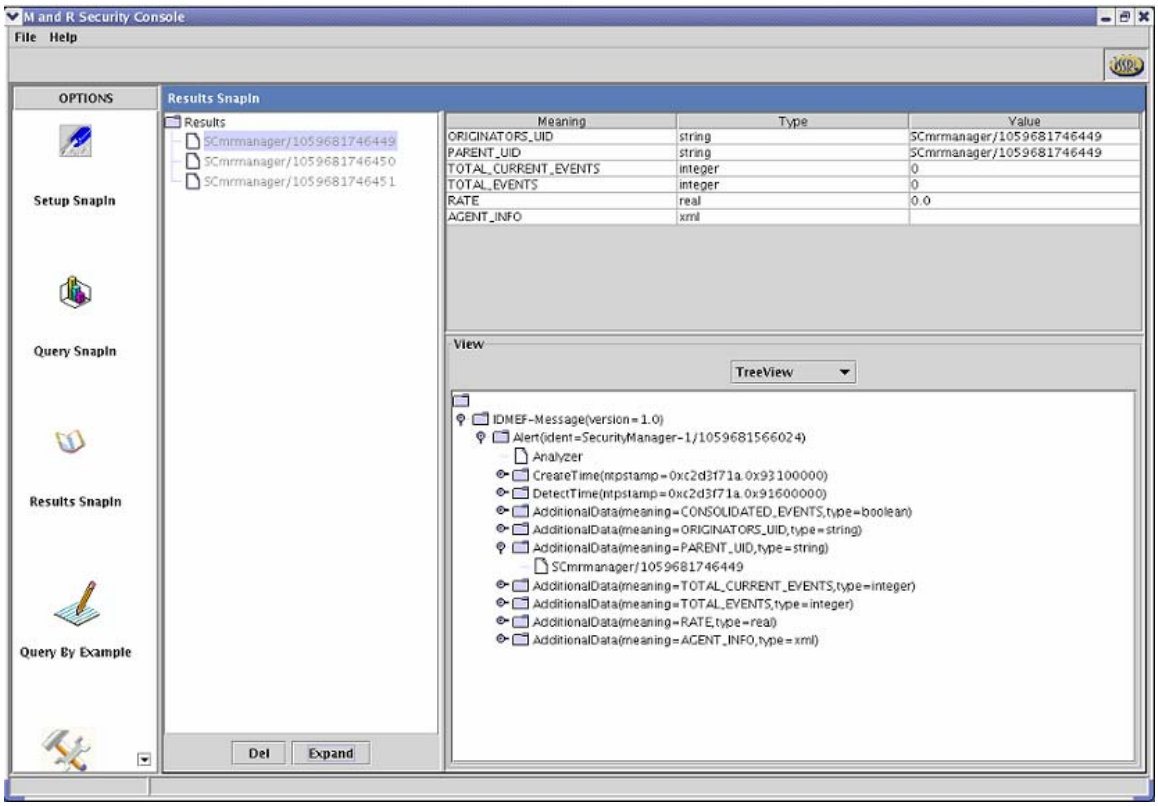


Figure 24: Results SnapIn displaying the results in high level without further expansion

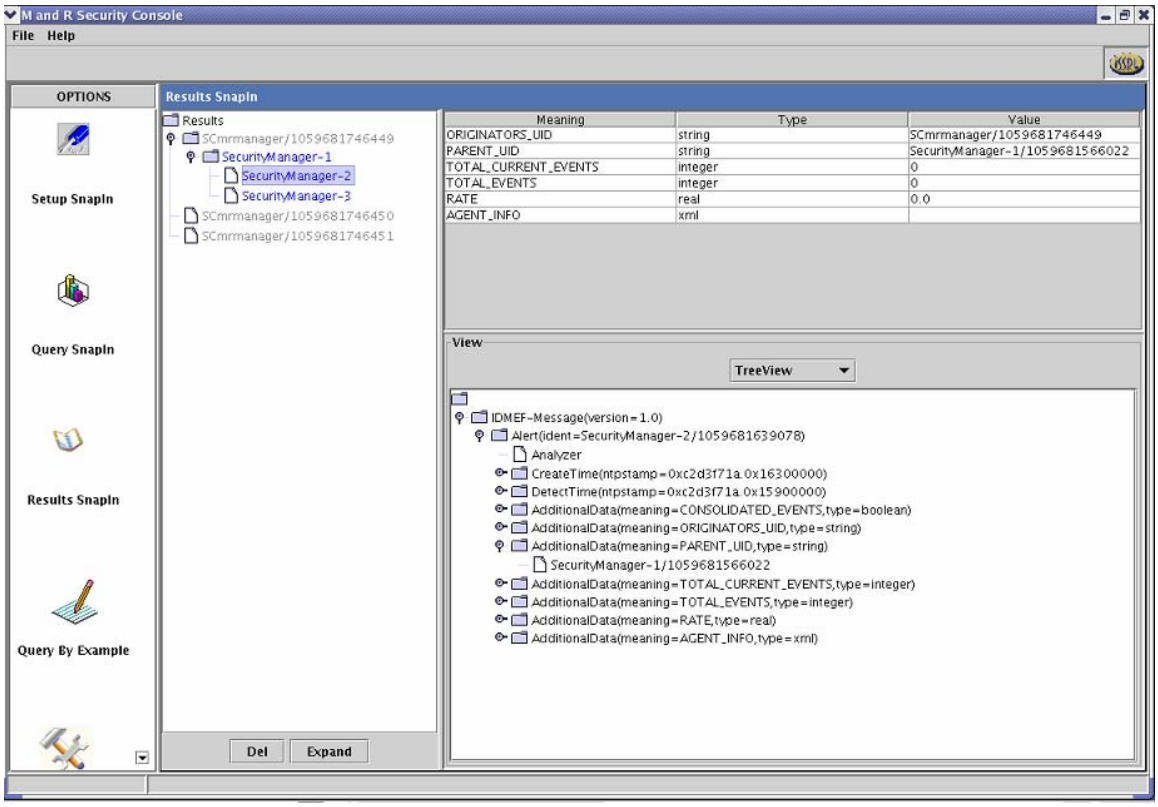


Figure 25: Display of results when one of the security managers is expanded to its next level

4.2.4.1 Results Expansion and Deletion

It is possible to expand the obtained results, if the results are aggregated. The expansion of the high level result generates the middle level results. A snapshot of these two kinds of results is shown in Figure 22. In order to provide the user the ability to expand the results, the interface incorporates a button called *Expand*. By selecting the desired Security Manager (aggregated result), and clicking this button, the user can expand the result further. This feature is also enabled with a double click on the Security Manager. *Delete* button as the name suggests, deletes the selected result in the tree.

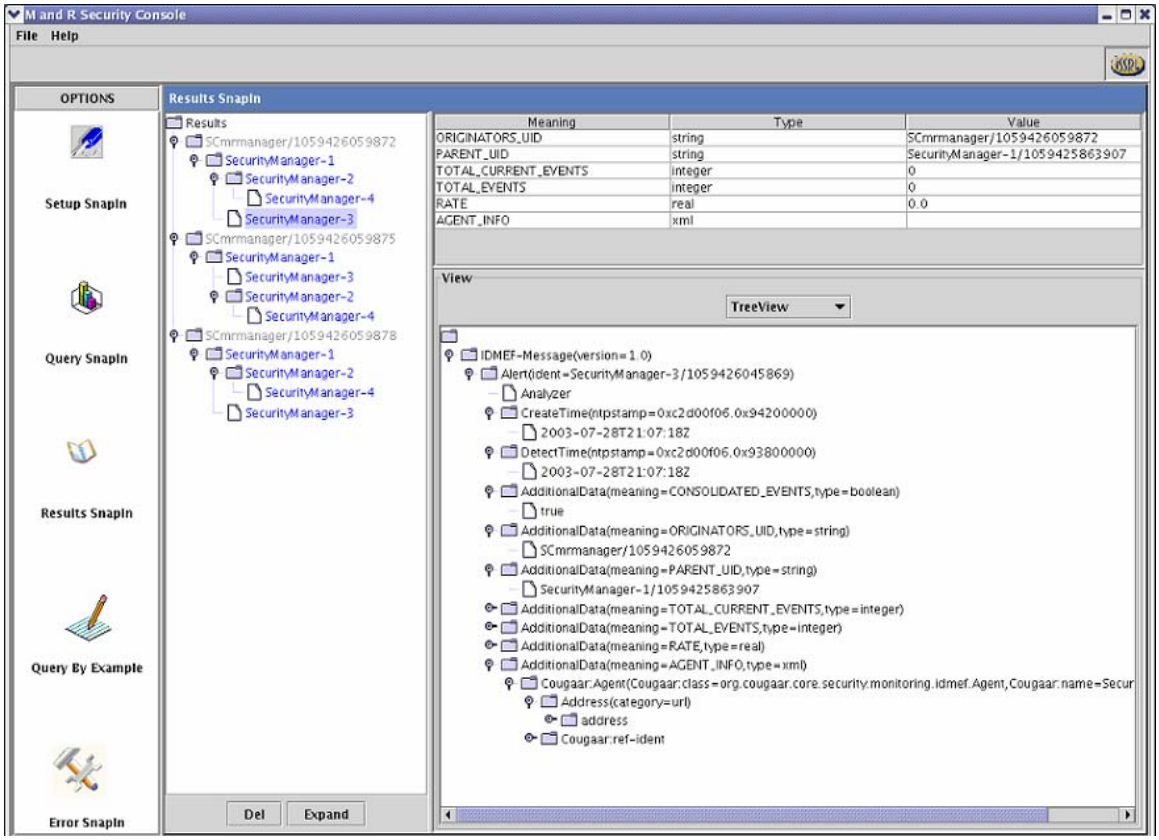


Figure 26: Results expanded to two levels.

Figure 23 shows the expansion of a Security Manager to its lowest level. As it is a Non-Aggregated result, the tabular format displays the several other attributes as shown in Figure 23. These attributes are the parameters of the corresponding IDMEF Object of the Non-Aggregated result.

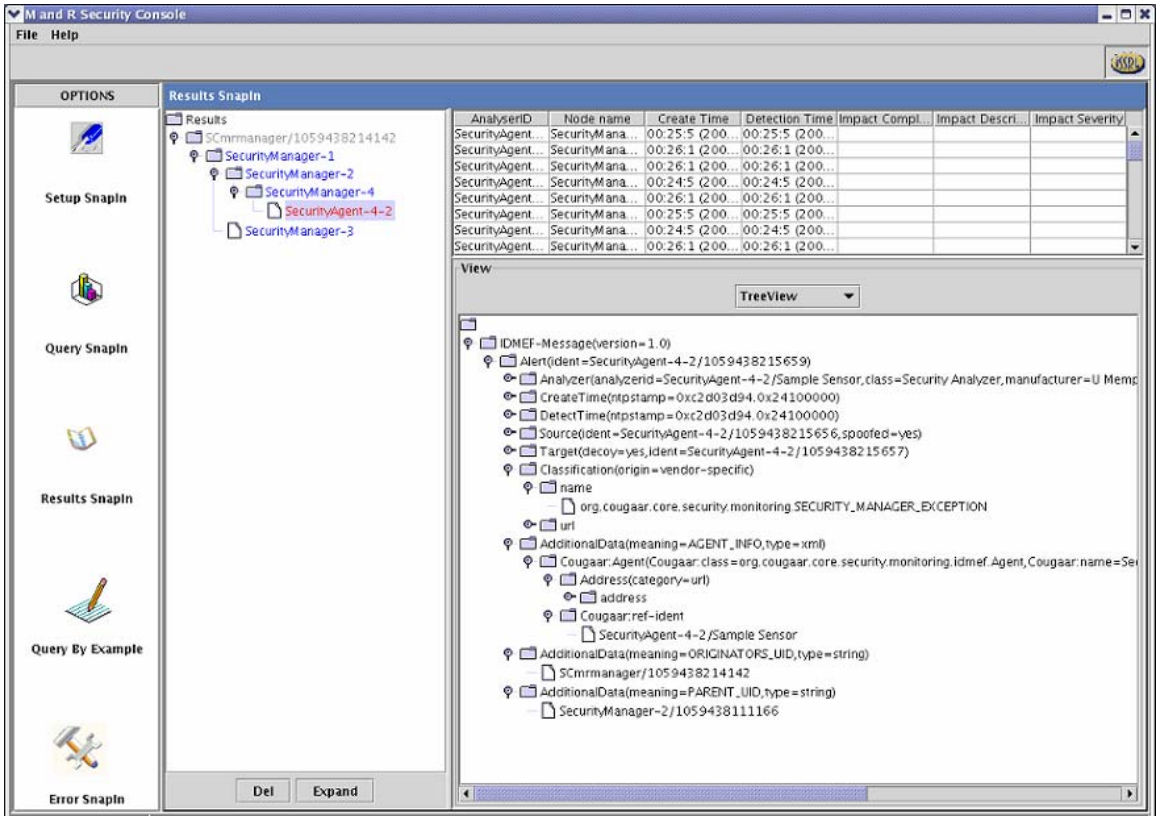


Figure 27: Results of a non aggregated query.

4.2.4.2 Visualizing Query Results

A click over the query name in the list displays the results in tabular format.

For better analysis of the results, three different view modes are provided to the user. An object can be selected and can be viewed in any of the three views: ***Text View, Tree View*** and ***Time Series View***. Time series view is disabled, as the results are IDMEF objects. Selecting an IDMEF object and selecting a view from the dropdown list of views displays that object in the selected view style. Sample snapshot of Text view is shown in Figure 24. TreeView shows the IDMEF object in Tree format as shown in Figure 20, 21, 22, 23. The user can shrink or expand any part of the tree (i.e. IDMEF object) to view the specific details.

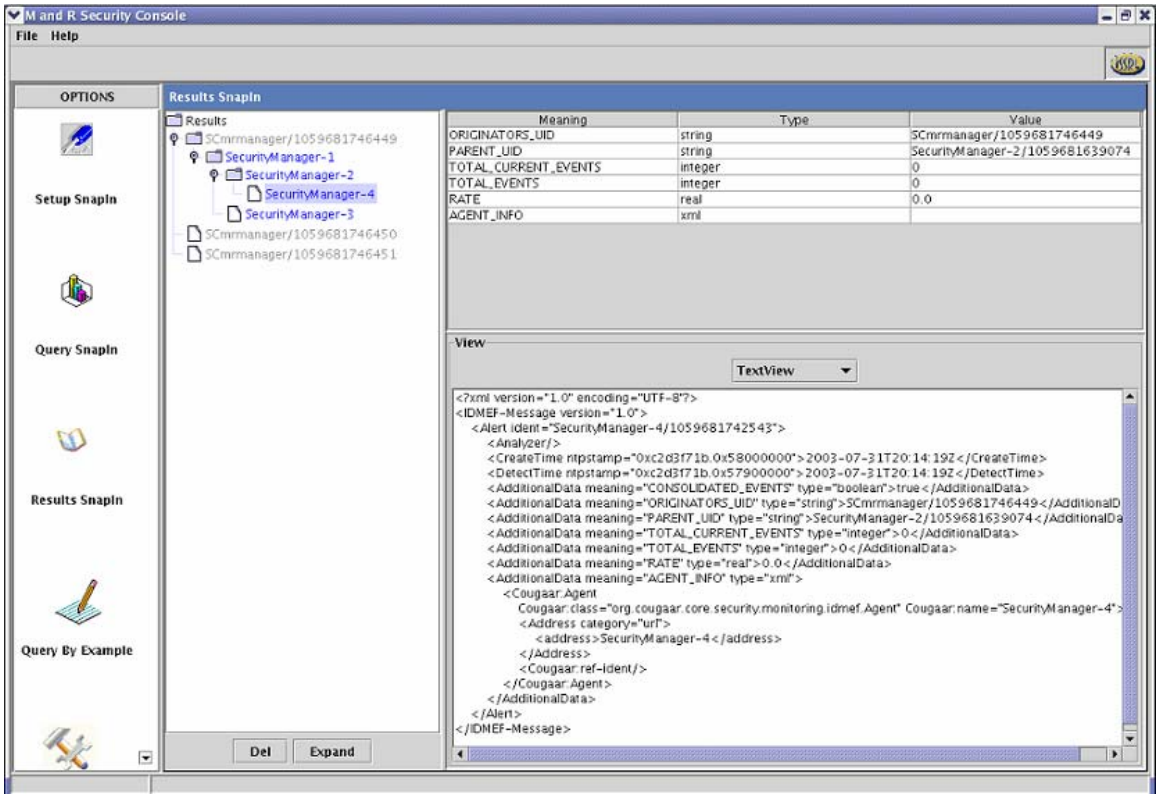


Figure 28: Results for a Security Agent in Text View

4.2.5 Query By Example

4.2.5.1 Description

Query by example can be used to build queries.

4.2.5.2 Defining Attribute value pairs and building query

For building a query an attribute of Alert object has to be selected, depending on the attribute its default values and comparison operators are displayed. After selecting an attribute, the appropriate comparison operator should be selected and the value entered. Once the value entered and the ‘Add’ button is pressed, the attribute is added to the query. If the user feels that the values are not required or incorrect, they can be removed by pressing 'Delete' button. After having all the required attributes, query can be built. Finally the query should be given a name. By pressing the 'Save' button query is sent to Query SnapIn, it is displayed in the ‘to be saved’ queries list. Figure 25 shows the interface for building query using Query By Example.

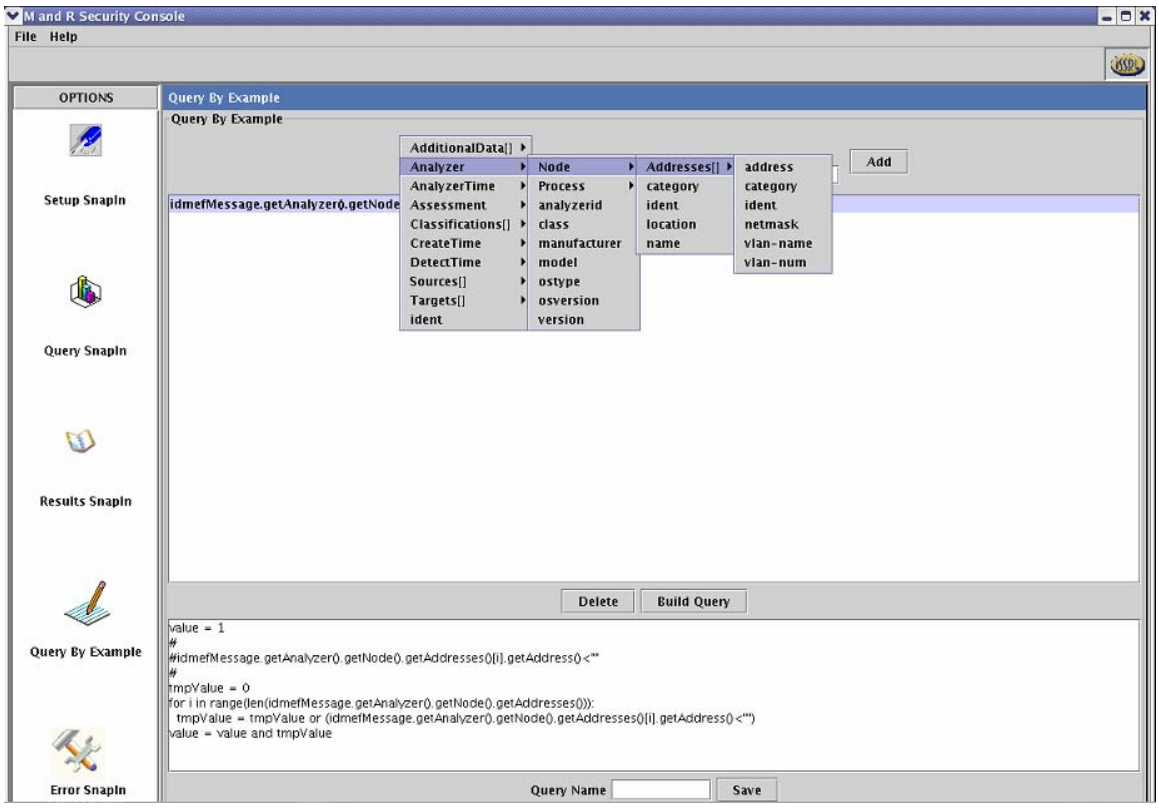


Figure 29: Interface for building query using Query By Example

4.2.6 Log SnapIn

Log SnapIn shows two types of Logs. One is ***Command Log*** while the other is ***Error Log***.

Command Log logs the messages while the connection is being made to the Society including *GetAll, Get-CM-Info, Get-MnR-Info, Open, Close.*

The ***Error log*** is deemed necessary as to make a log of the possible errors that might result when the user tries to connect to a security console manager. These errors are could be the delays caused in connecting to the security console manager. It could also be the absence of the SCM being queried for Expansion or Deletion. An instance of few recorded errors are displayed in Figure 27. As can be seen from the figure, each error displays the date and time of the error occurance along with the name of the security console manager which caused that error. For example, reported is the Delete operation failing on the security console manager over the aggregated query as shown in Figure 19.

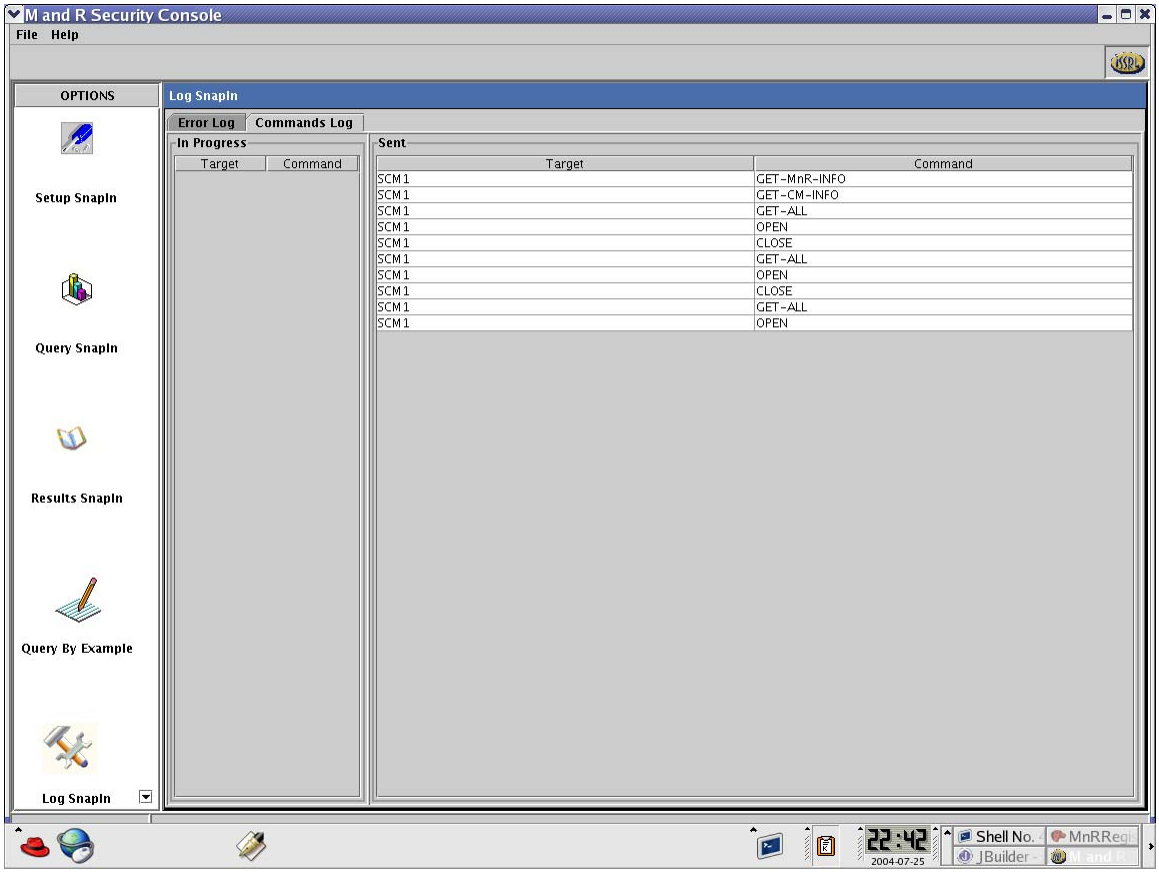


Figure 30: Log SnapIn showing the Commands Log with logs of commands.

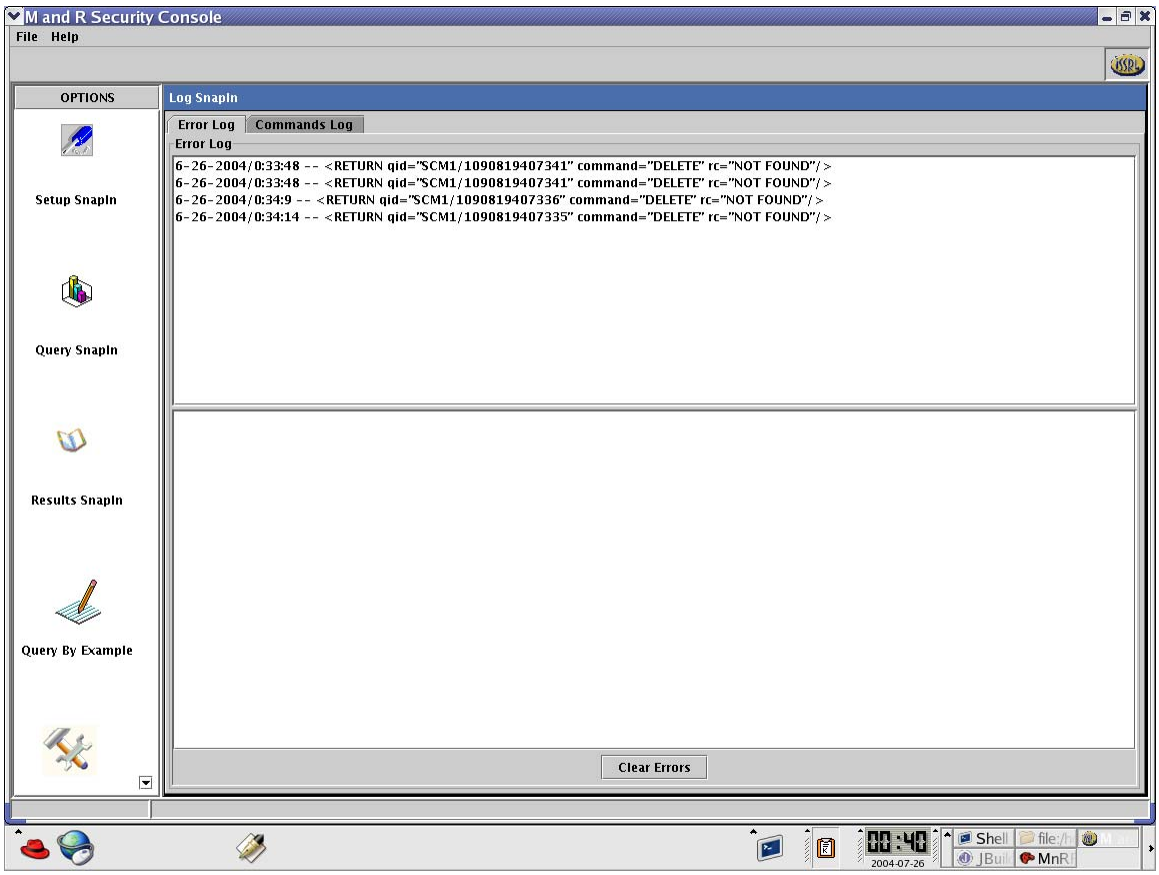


Figure 31: Log SnapIn showing the Error log with some log of errors

4.2.7 Mining SnapIn

This SnapIn requires that the XML Database (eXist) is configured properly for retrieval, storage and manipulating off-line messages. This SnapIn requires that the incoming Messages for any query(s) in the Results SnapIn are extracted into the database or the log file.

This SnapIn provides the basic User Interface for carrying out the entire knowledge mining over XML messages (that are in IDMEF format). The SnapIn contains Four related Modules for realizing the complete Mining and Analysis phase including the database management.

4.2.7.1 Description

Database Management: The Primary module for interfacing with the XML Database and supporting the execution of commands for database updating, deletion and insertion in a special form. This module also communicates with the GUI Component for assisting the user with Views and messages as appropriate. The user has the facility to upload the database from the Log file too. Basic Database handling commands are provided. It provides the high-level view of the database in a tabular form. The interface provides the options to select the records in the database. These selected records can be deleted from the database or can be sent to the next interface to carry out the statistics phase.

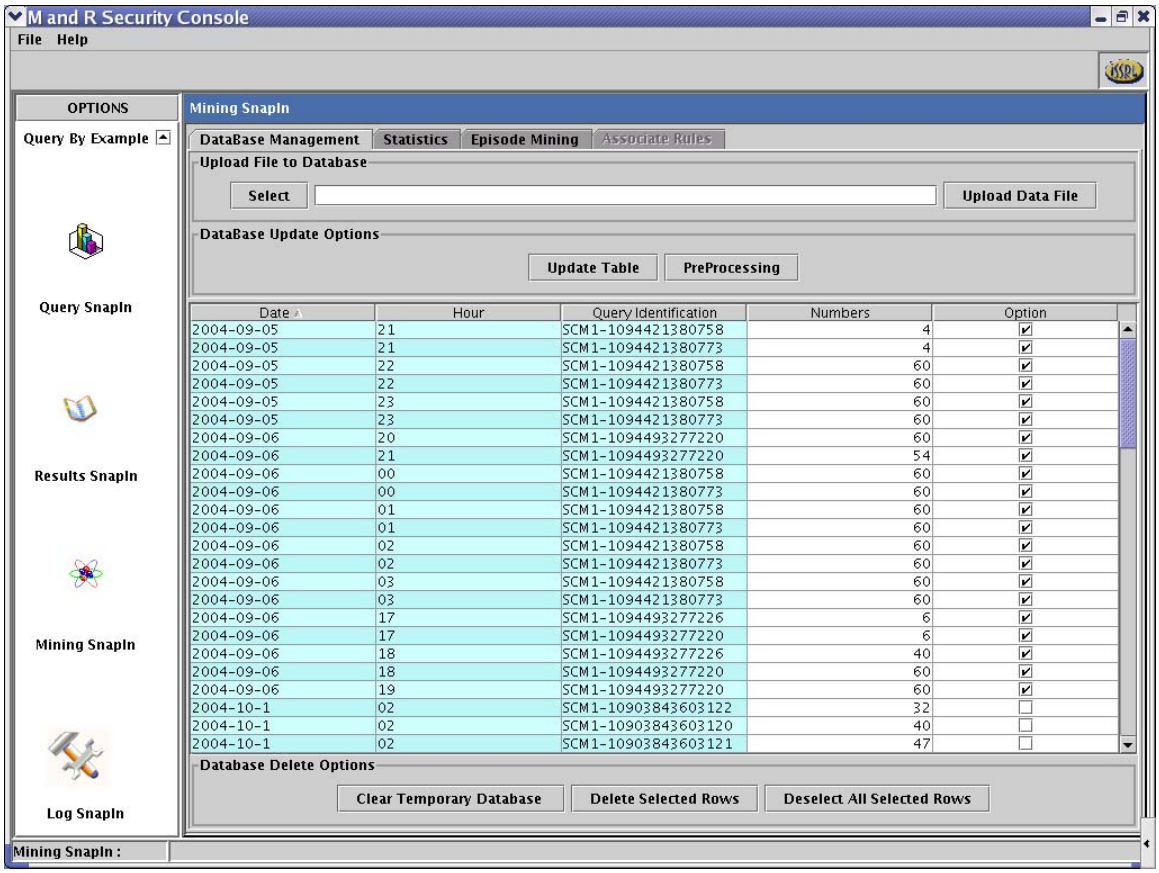


Figure 32: Starting Database screen with all database Management controls.

Statistics: This interface provides the user with the most crucial Statistics of the messages and their contents in a graphical representation. After the user has selected the records from the database to analyze, this interface is loaded with the queries to perform data-analyses. The results are shown in two views. The XML result is shown in the hierarchical tree view (using JTree representation) and chart view using the chart API. There are two charts for providing the detail view of the result. The top chart provides the basic analyses. When the user selects any data represented in the chart, its specific details are further elaborated and depicted in the bottom chart. Thus, the top chart provides a high level view while the bottom chart shows the detailed view of the database query result. It is user friendly, in as much as there are options to modify the view, zoom along x and y axes and print the view for distribution or record keeping.

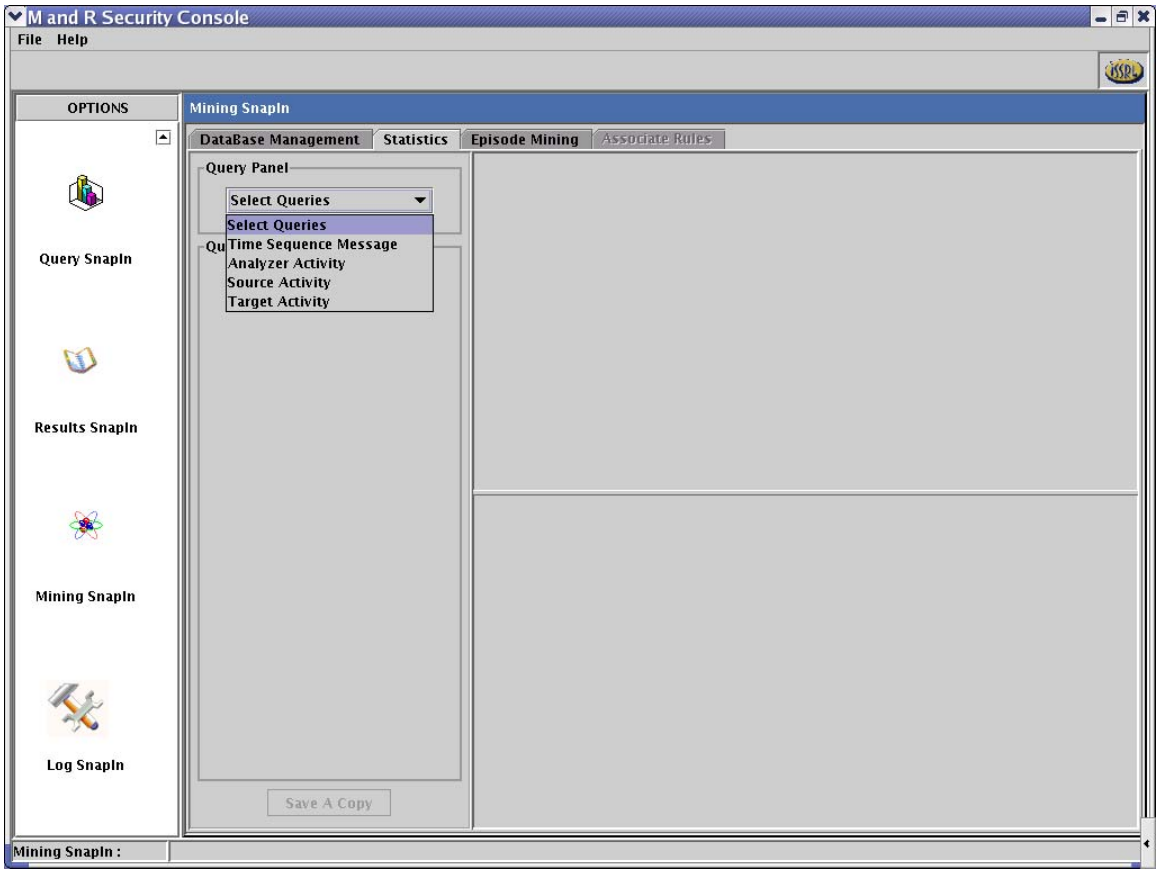


Figure 33: Statistics screen showing the List of available queries for generate selected message statistics.

Episode Mining: The third module of this Snap In, also the starting point of initiating the Mining process is associated with the concept of stream data mining, in this case Frequent Episode Mining. This module incorporates the process of recursive processing of messages and reorganizing the related events in the messages as candidate Episodes, under the given parameters. The bulk result is regrouped and portrayed as a suitable View. This is communicated back to the GUI component and supports multiple Visuals to mark the density and the relationship for each event. The module is fully functional for Serial as well as Parallel mining sequences.

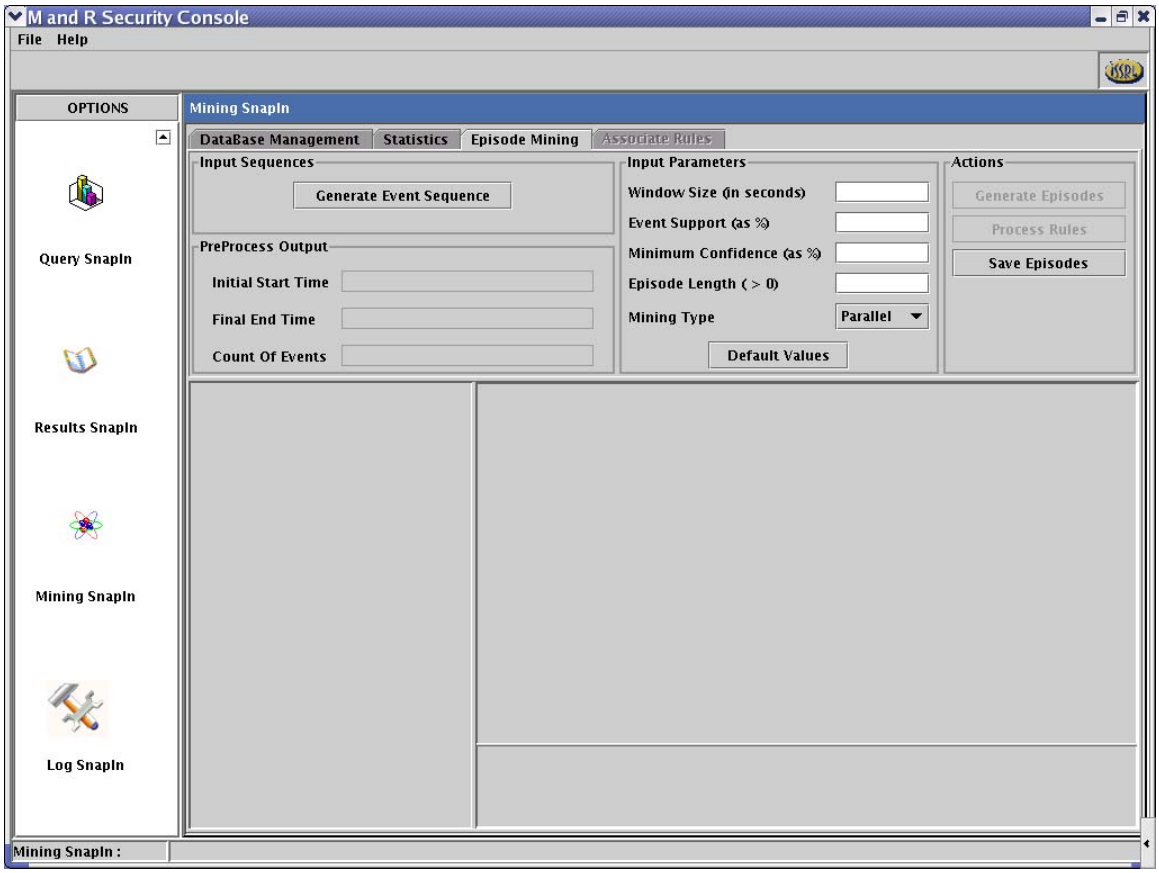


Figure 34: Episode Mining screen for generating event sequences, generating candidate episodes and visualization.

Associate Rules: The final module in the Mining Snap In incorporates the ability to portray the relations among the frequent episodes as Rules. The module communicates with the GUI component for visualizing the Rules in an appropriate fashion. The Rules have the notion of Right hand and Left hand sides designated as antecedents and consequents. Either and/or both the antecedents and the consequents can have multiple events. Hence, the Antecedent can be of length one, two or more while the Consequent can be of length one, two or more too. SC depicts the Rules and the Visualization shows them in a graphical form along with Support and Confidence measure for each Rule generated.

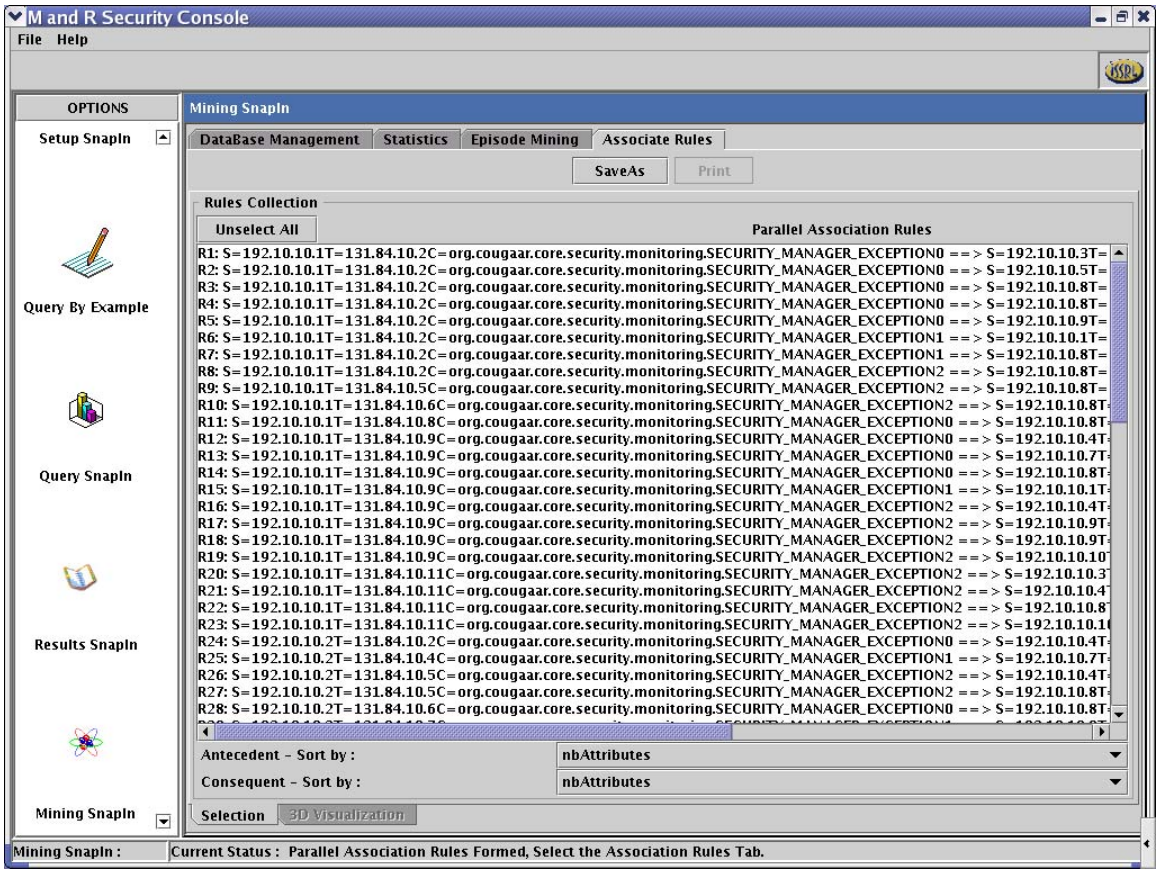


Figure 35: Associate Rules screen showing the Collection of Rules generated after the candidate episodes of Mining were related with each other. These are Parallel Association Rules.

4.2.7.2 Detailed Description – Database Management Module

The Interface supports manipulation of database records stored as collection. The main Visual is the JTable that provides a high level view of the entire data base collections. The two snapshots are shown below to depict the functionality.

The Interface is divided into Four major sections – *Upload File to Database*, *Database Update Options*, *table*, Database *Delete Options*. Illustration, Figure 32.

Upload File to Database – This incorporates selection and Uploading the Data File. It allows uploading files having xml messages only. Two file extensions are currently allowed, *.xml* and *.log*. User can open the saved log file for updating the database. The Upload Data File is a command to the DataBase Manager to upload the contents of the log file into the Data Base. Proper care is taken to avoid the redundant and null values.

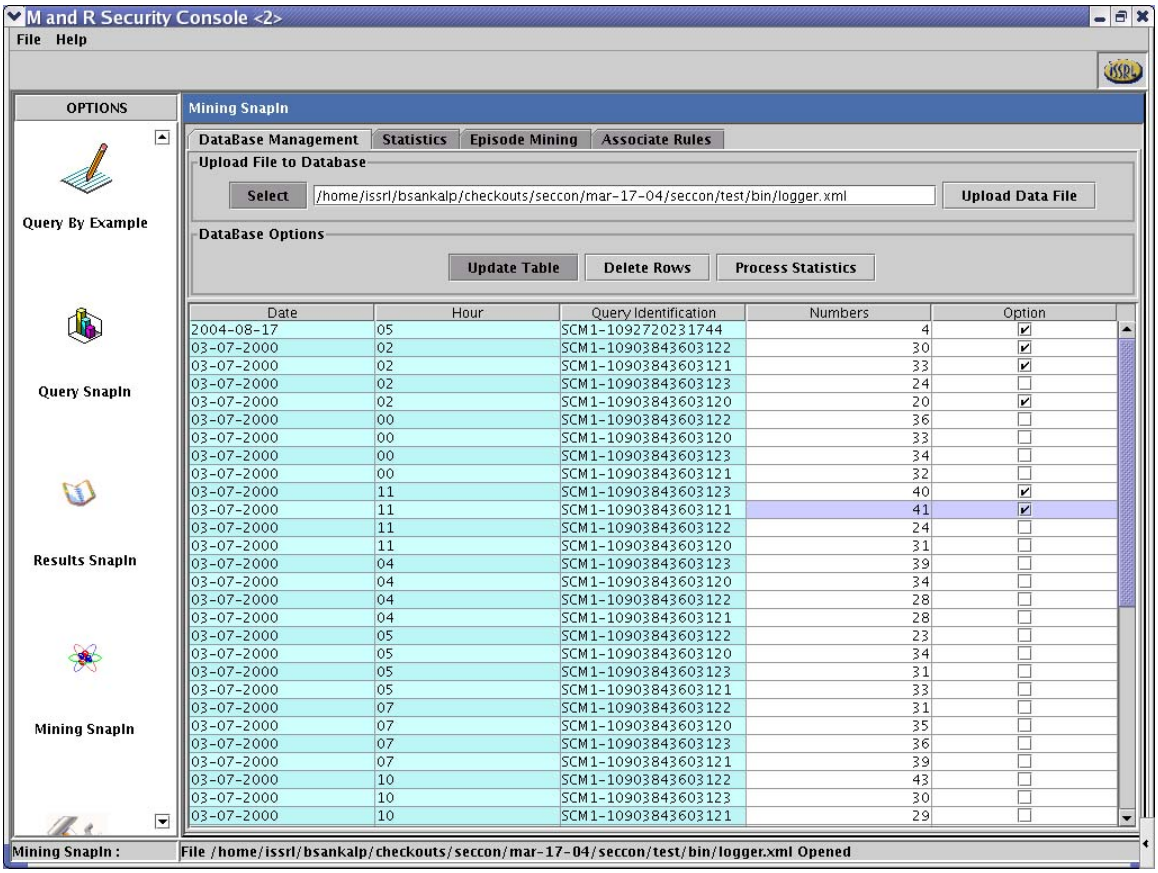


Figure 36: The table showing the message collection format. Also shown is a log file, logger.xml for uploading the Database Collections Repository.

DataBase Update Options – The table visualizing the internal database contents can be upgraded with selected records for analyzing (Statistics and Mining) purposes. To achieve this, there are two buttons provided, Update Table and PreProcessing. Update Table is like refreshing the contents whatsoever, as and when the user decides. PreProcessing is for extracting a specific sequence of messages as per user selection from the database table for further analysis.

Table – This is basically used to provide a high level view of the xml-messages data collected and stored in the database. The records are organized into four columns, Date, Hour, QueryIdentification, and Numbers. The last column is for the options that enable the user to selectively select or deselect the rows from the table before doing any further processing. The individual columns support sorting in either order. This is to assist user in grouping messages close to each other in time and then do the required analysis (statistics and/or mining).

Database Delete Options – This section assists maintainability of the data base. The old, unwanted or unsuitable data records can be discarded at the user's discretion. To achieve this, there are two buttons provided, Clear Temporary Database, Delete Selected Rows. Clear Temporary Database is for completely erasing the records in the temporary database unit. This unit was created during the preprocessing phase for concentrating on selected records for analysis by other modules (Statistics, Mining, Rules). Delete Selected Rows is for permanently erasing the records from the database corresponding to the selected rows from the table.

4.2.7.3 Detailed Description – Statistics module

This module provides the Visuals to perceive the message form, its content and the relationship with time (Time Sequence chart). With the incorporation of Chart views the distribution of the IDMEF message parameters and their inter-relationships are presented for getting the statistics details. Four major attributes of the message content are taken into consideration for analysis. These are ***Time Sequence Messages***, ***Analyzer Activity***, ***Source Activity*** and ***Target Activity***.

This module communicates with the *XML Query Component* for retrieving Query result for each Query sent to the database. There are two views provided for Visualization, Tree View and Chart View. The details follow with illustration, Figure 33.

The Interface is divided into three sections – *Query Panel*, *Query Result*, *Chart View*.

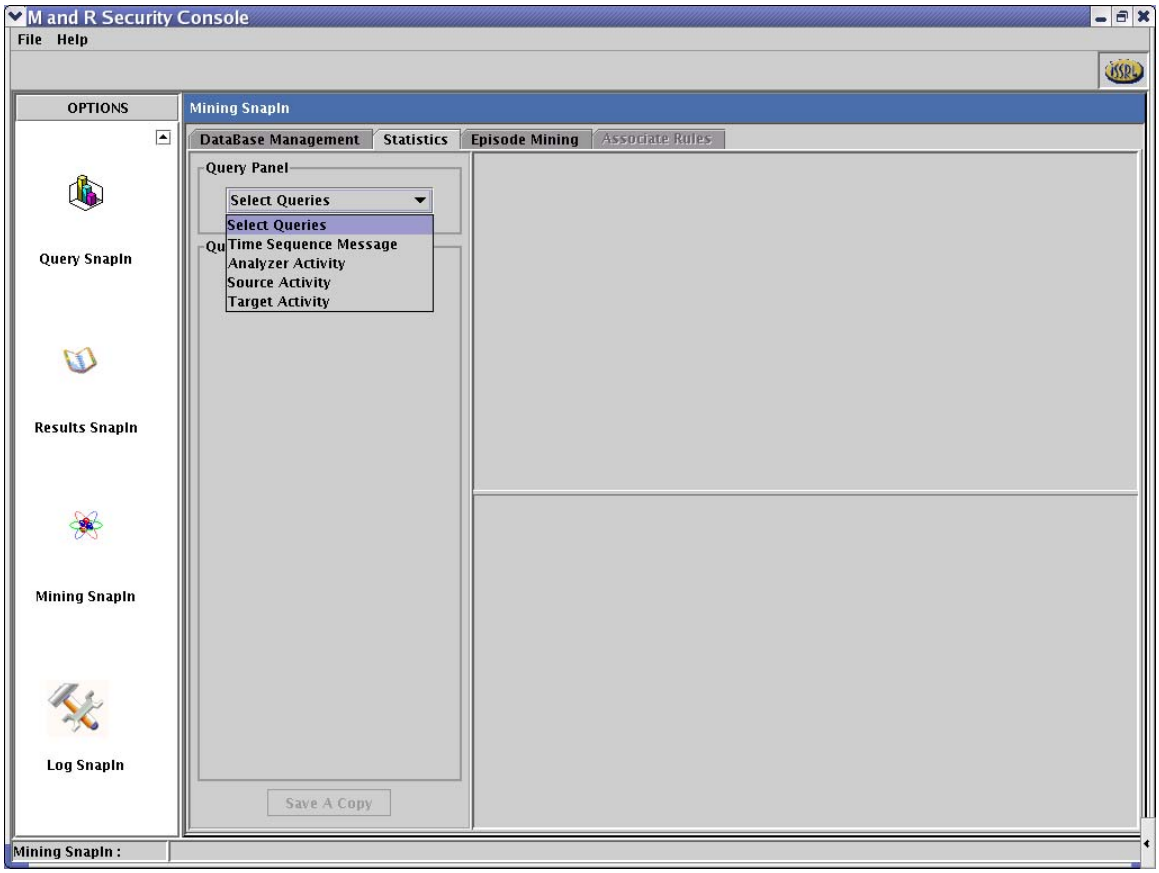


Figure 37: Statistics SnapIn – The list of Queries for generating the detailed Statistics.

Query Panel – There are four categories of queries that are handled by this Section. ***Time Sequence Message***, ***Analyzer Activity***, ***Source Activity*** and ***Target Activity***. For each Query sent, the user has the option to save a copy of the result.

- ***Time Sequence message*** shows the high level distribution of messages over time. These are the messages that the user desired to do analysis upon. It represents the spread of the Alert messages that were judiciously selected by the User from the Collection of messages from the database over a certain period of time. Illustration, Figure 34.

- ***Analyzer Activity*** depicts the detection of alerts in a period of time. The identification of the Security Analyzer and the count of the number of messages for each are shown in the upper chart. The time sequence in the lower chart shows the time distribution of detection for each individual Analyzer. Illustration, Figure 35.
- ***Source Activity*** depicts the source(s) of attack(s) in a period of time. The identification of the Source Node and the count of the number of attacks perpetrated by it are shown in the upper chart. The time sequence in the lower chart shows the time distribution of the numbers for each individual Source Node. Illustration, Figure 36.
- ***Target Activity*** depicts all the target(s) as retrieved from the incoming message, in the given period of time. The identification of the Target Node and the count of the number of attacks made on it are shown in the upper chart. The time sequence in the lower chart shows the time distribution of the numbers for each individual Target Node. Illustration, Figure 37.

Query Result – The XML Query Component provides the result details in XML form. The suitable View for this result is a hierarchical tree. This view is provided using the JTree component of Java. It responds to the user selection and appropriately loads the chart concerned.

Chart View – This provides the high level and a detail level view by using two charts. Except the Time Sequence Message Query results, all the other results are shown in two view levels. The upper Chart showing the distribution, while the lower chart shows the details for each distributed element as a sequence in time. For the Time Sequence Message the distribution involves the sequence in time with their count specified.

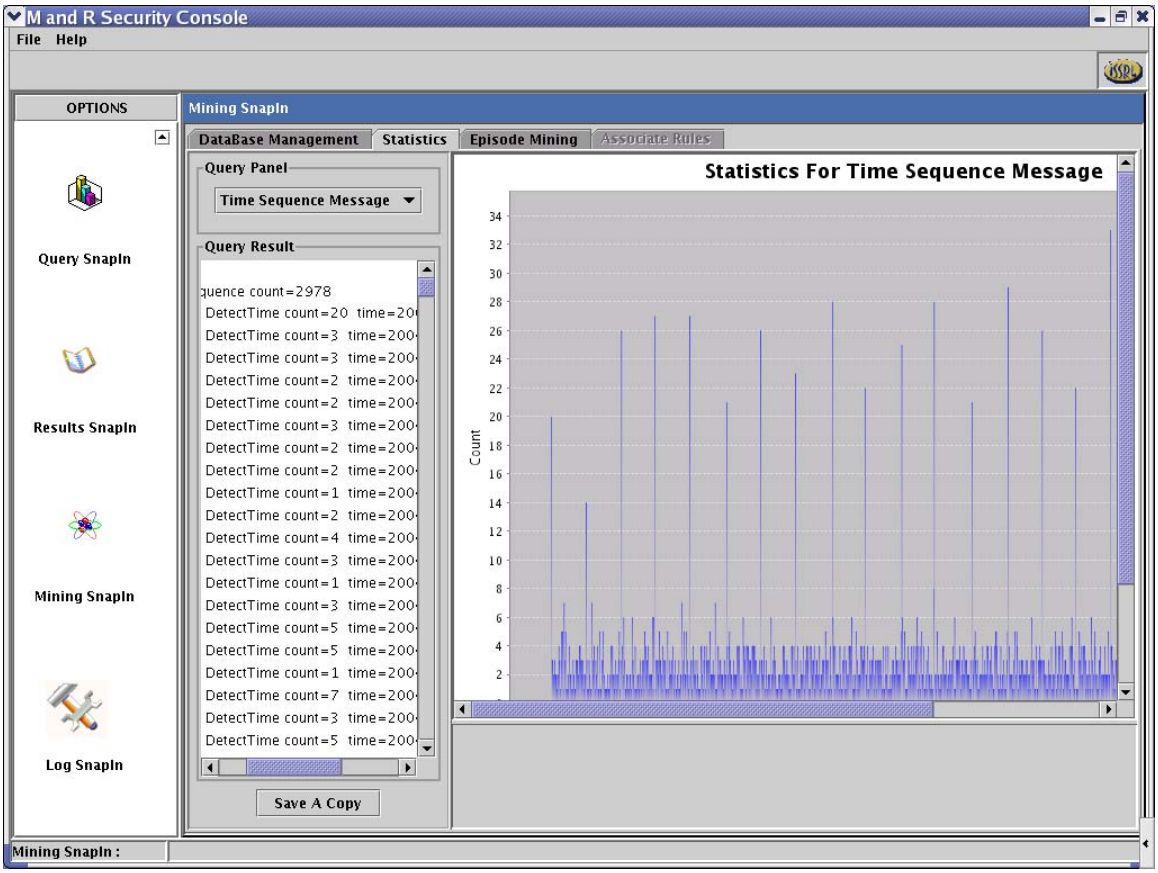


Figure 38: Statistics of Time Sequence message showing the Query Result on the left ScrollBar depicting the DetectTime, and the count, also the total sequence count.

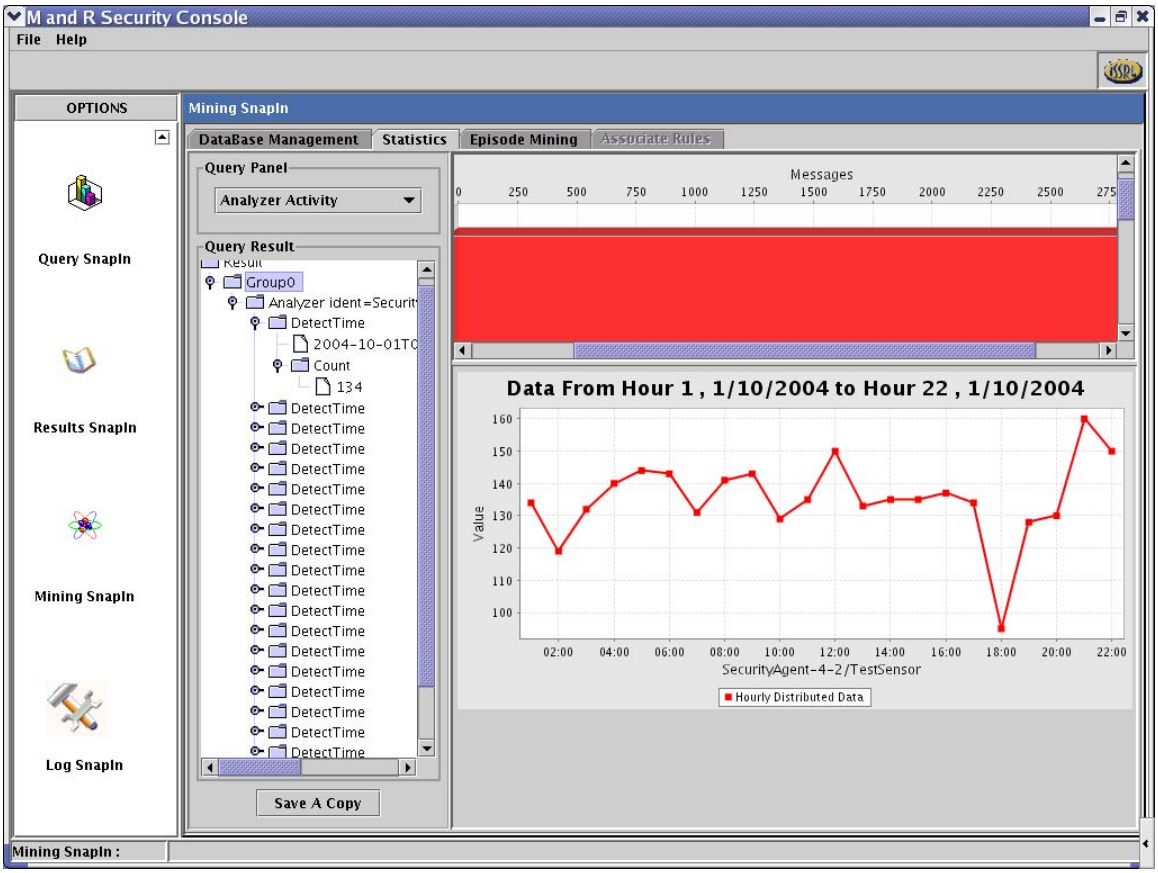


Figure 39: Statistics of Analyzer Activity provided with the tree view on the left and chart views on the right. Tree View details the hierarchy. The upper chart shows the messages for the series of Analyzers and the lower chart shows the time sequence distribution for the Test Sensor, SecurityAgent-4-2

Figure 40: Statistics for Source Activity. Left hand ScrollBar shows the detailed tree View. The upper chart shows the Message count distribution for each Source while the lower chart depicts the time sequence for Source Node 192.10.10.10 for the dates.

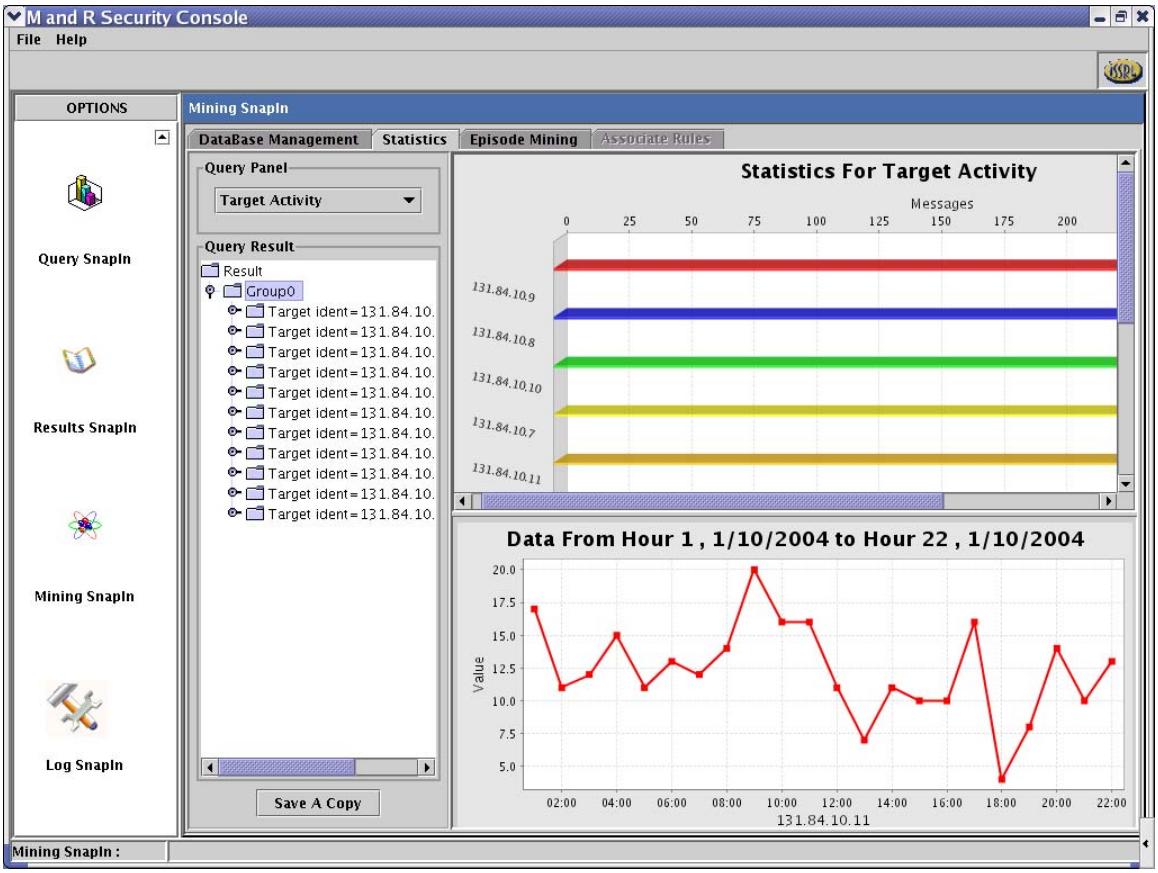


Figure 41: Statistics for Target Activity. Left hand ScrollBar shows the detailed tree View. The upper chart shows the Message count distribution for each Source while the lower chart depicts the time sequence for Source Node 192.10.10.10 for the dates

4.2.7.4 Detailed Description – Episode Mining module

This module deals with the discovery of frequent episodes from the selected messages that have events occurring with some frequency and have certain order of relationship among them over a period of time.

The module proceeds to work in a sequence. The order is important to achieve the actual result. The module enforces the order. First, the Messages for Analysis have to be *PreProcessed* for generating Event Sequences. This initializes the algorithm for Episodes Mining. The next step is to provide the *Mining Input parameters* for emphasizing the conditions and criteria for Mining. Here the user has to judiciously provide near accurate parameters to get good result. Here the user also provides the Mining Type that is desired (Serial or Parallel). It has to be noted that the Association Rules that are generated in subsequent steps also depend on the mining type selected. Illustration, Figure 31.

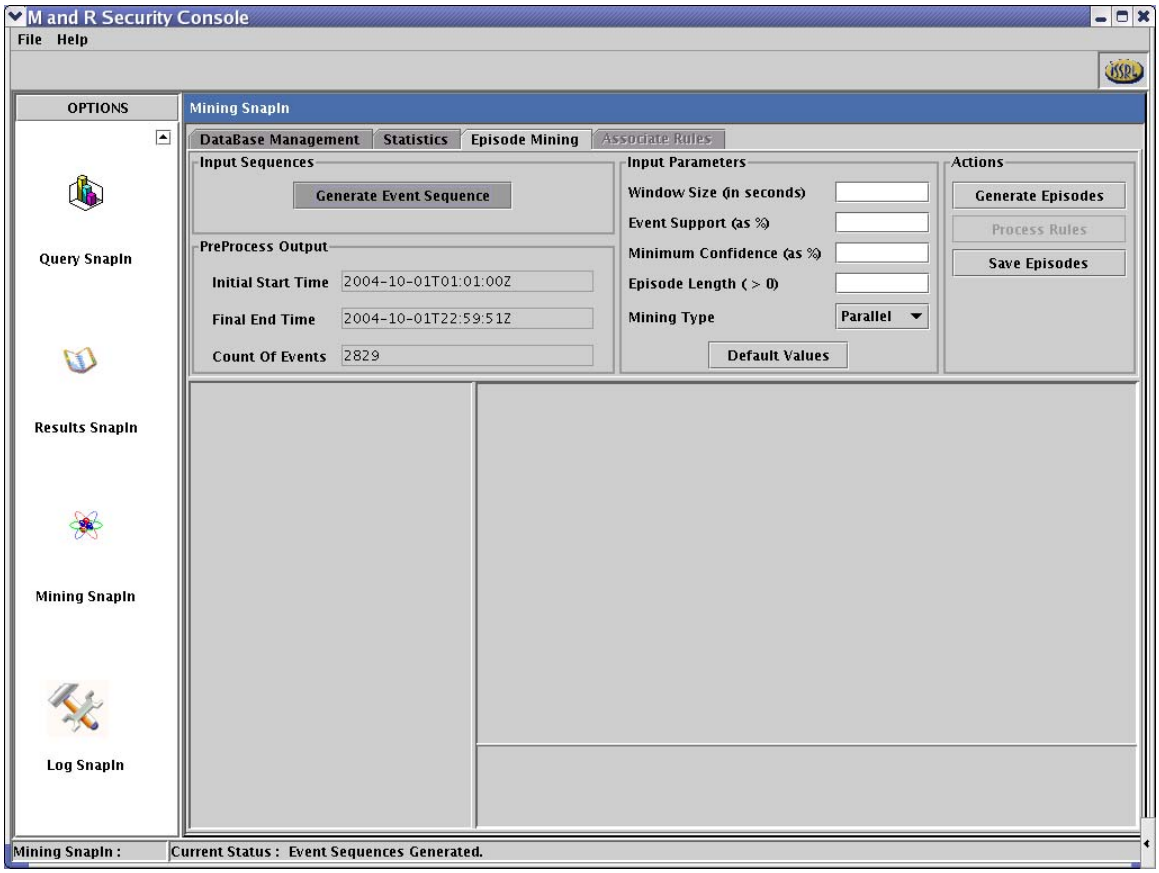


Figure 42: Episode Mining showing the Generated event Sequence Output.

The results of the frequent episodes generated, is viewable as Tree, Chart and Graph. This result can be saved in XML format for any future reference. The hierarchical relationship in the tree shows the top three-level details of Episode Result. Episode Length, Episode Group, Individual Episode, Episode event(s) in that order.

Preprocess: - The event sequences are generated and a brief statistics is output on the Console - **Start-time, End-time** and **events count**.

Input Parameters: - The Mining algorithm requires five formal parameters from the user to start the mining process.

- ***Window Size*** – The values here must be Numeric (>0) and it represents the size of the sliding window in seconds.
- ***Event Support*** – The values here must be Numeric and it represents the threshold for an Episode which scans individual events. As the values are to be specified as a percentage, the range of input should be within 0 – 100.
- ***Episode Length*** – The maximum length of Episodes desired is specified here as a positive Integral Number (>0).
- ***Mining Type*** – The type of Mining desired, could be either serial or parallel.

- ***Minimum Confidence*** – The values must be Numeric (>0) and it represents the confidence in the Rules generated and provides the basis of selection from among the bunch of Rules generated. This parameter is basically for Association Rule module and not a direct result of the Mining module.

Episode Generation – For generating the Frequent Episodes based on the input parameters. The result is provided as three different Views. Tree View is for showing the higher level hierarchical order, Chart for showing the statistics of episodes and their numbers for each Episode Length. The graph view provides the relationship among various events of an Episode. This is the lowest level detail, showing the details of each event's contents.

Process Rules – Once the Episodes have been generated, the rules are formed that involves a relationship between the set of events of the participating episodes. Each event possesses attributes namely, Source, Target and Classification. The rules generated are handled in detail by the Associate Rule module.

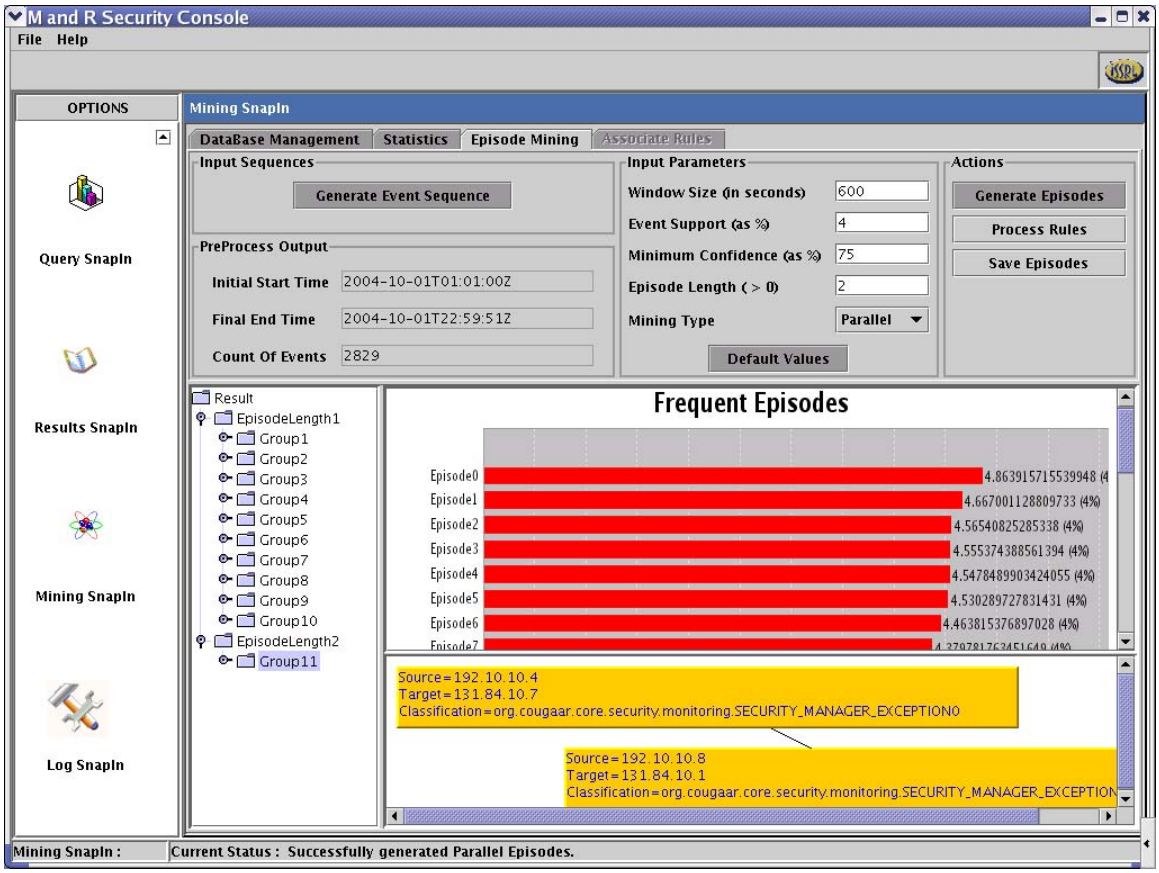


*Figure 43: Episode Mining showing the Frequent Episodes for the **Parallel Mining**. The tree View shows the groups for each Length Type. The upper char shows the count of each Episode and its percentage distribution. The lower chart depicts the two events for the length two Episode and the event details.*

4.2.7.5 Detailed Description – Associate Rules module

This module is responsible for providing accumulation of the rules and criteria for each rule in two different Views, List View and 3D View. The Rules are numbered and are represented as R1,R2 .. and so on. Each Rule is formed from its Right hand side and Left hand side. The Left Hand side forms the ***Antecedent*** of the Rule while the Right Hand side forms the ***Consequent*** of the Rule. There are two criteria for each Rule - ***Support*** and ***Confidence***. ***Support*** deals with the threshold for the event whereas the ***Confidence*** is associated with the Rule formation during the generation of Association Rules.

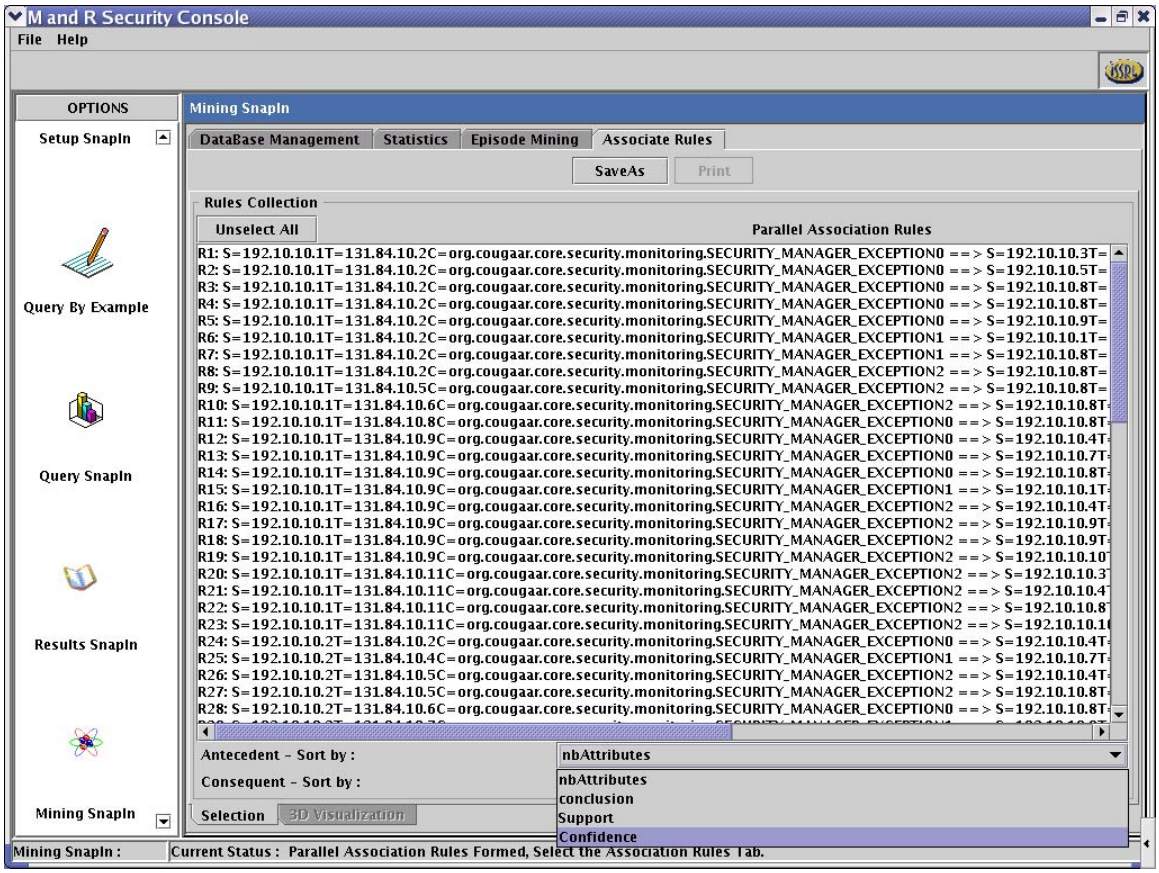


Figure 44: Associate Rules showing the Rules Collection and the Rules are Sorted by their Confidence.

A typical Rule is represented in this module as –

Antecedent ➔ Consequent (For Episode Length 1 on Antecedent as well as Consequent)

R11: S=192.10.10.1 T=131.84.10.2 C= Security_Manager_Exception0 ➔
S=192.10.10.3 T=141.84.10.1 C=Security_Manager_Exception2

R (N):- This Rule is identified by its number N.

S: - Source Node used for the Attack

T: - Target Node that the attack was aimed towards.

C: - Classification Type, that indicates the exact attack type used by the perpetrator.

This Module has two panels for visualizing the Rules.

Selection Panel: - This Panel has the List View that incorporates Sorted List Model. This custom model supports selection of multiple Rules and sorting of rules. There are four attributes on which the sorting is performed. ***nbAttributes***, ***conclusion***, ***support*** and ***confidence***. ***nbAttributes*** are the number of different event types in the generated rules. ***conclusion*** denotes the rule's consequent which contains one or more events. ***support*** is the threshold frequency and is also a criteria parameter for each rule. ***confidence*** is the rule's threshold value and is also a criteria parameter for each rule.

3D Visualization Panel: - This panel holds a Java 3D object. This panel provides a three dimensional view of the Rules and their relationship with event types, Support and confidence. The Antecedents are represented as Green cubes while the Consequents are the represented as Red cubes. The Support and Confidence parameters of each rule are shown as Orange and Blue colored cuboids. Also the range for the Support and Confidence are depicted for the selected rules in the View. Inherently, the 3D object has the facility to rotate in response to mouse movement (left button click and drag).

This panel provides a Save As option for saving the Image in a file of any desired format. The filters have not been defined for this purpose.

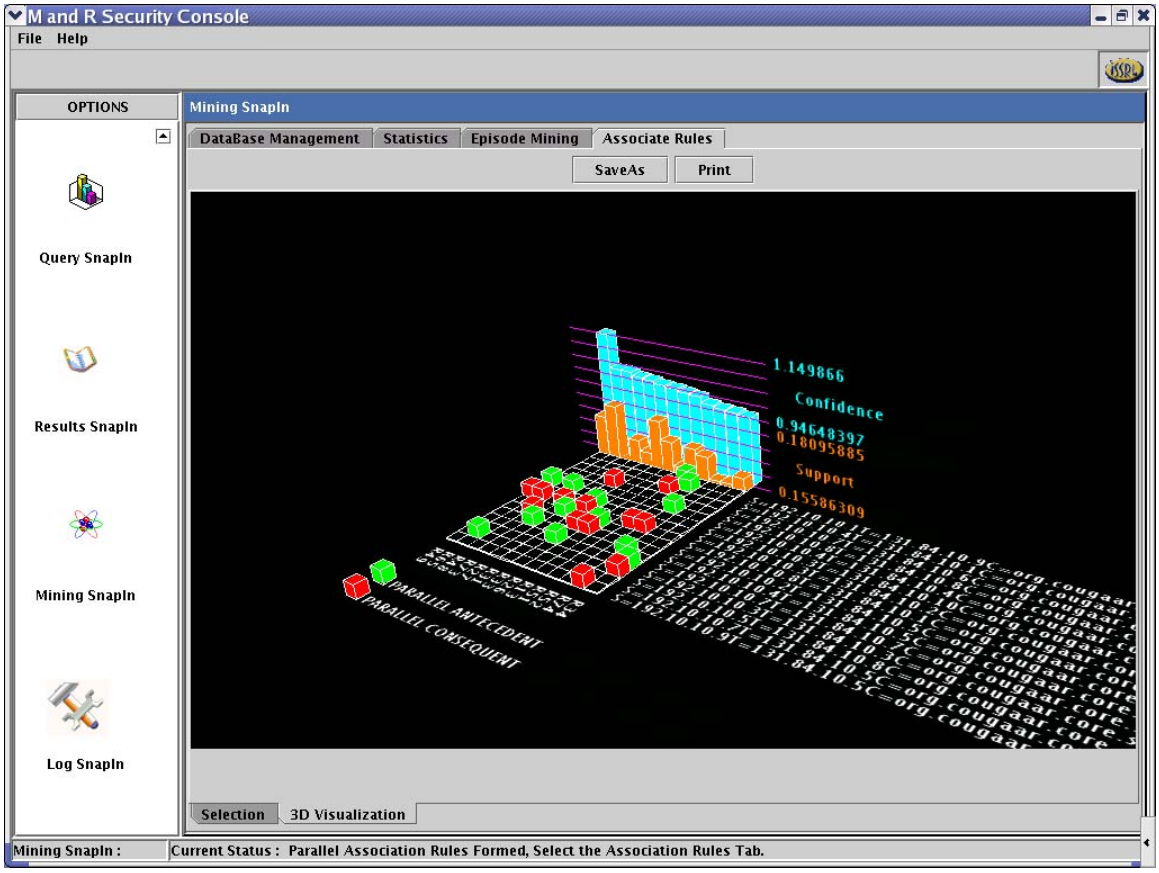


Figure 45: Three Dimensional Visualization of Association Rules

4.3 Security Console Packages

4.3.1 Seccon Package Classes

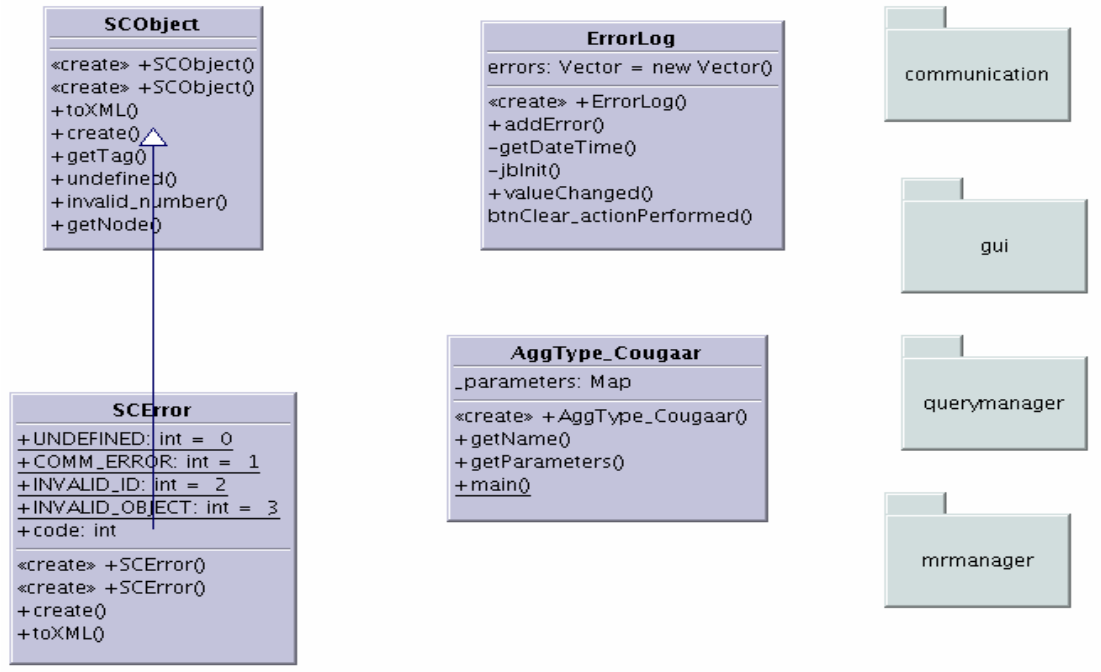


Figure 46: Classes in edu.memphis.issrl.seccon package

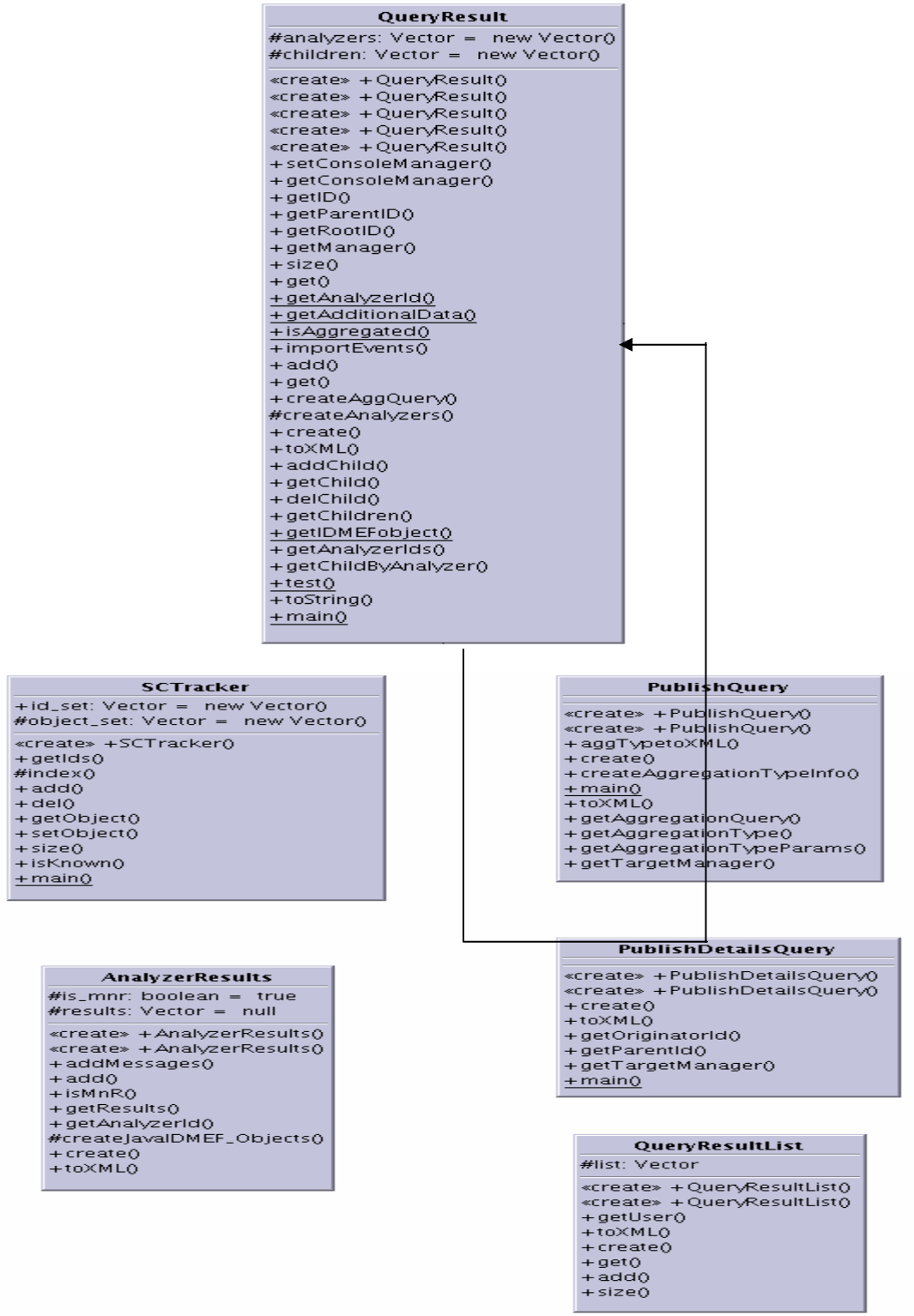
public class SCError extends SCObject
Error information.

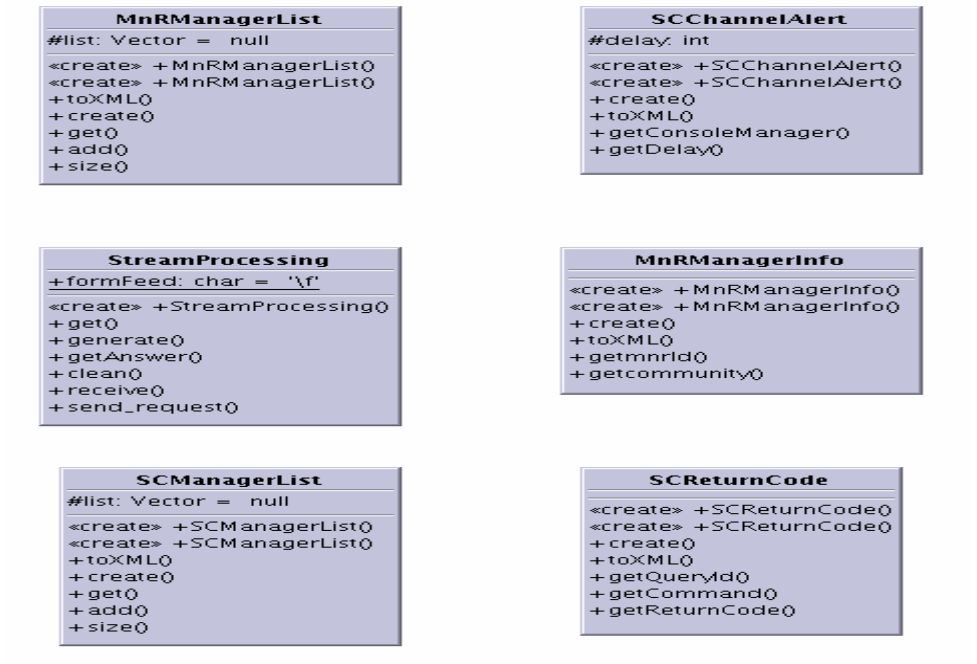
public class SCObject implements Serializable
Abstract class of objects processed by the Security Console.

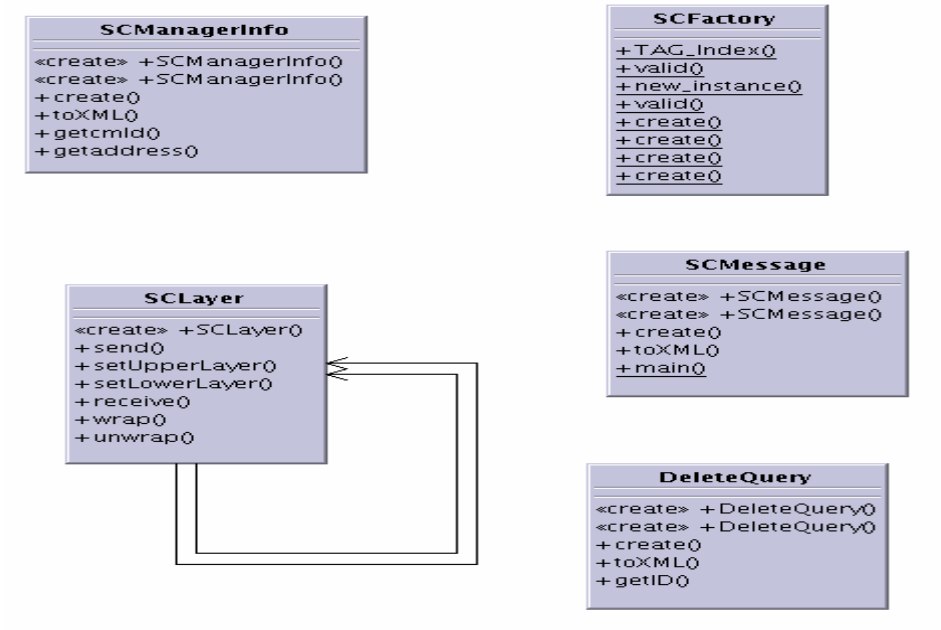
public class AggType_Cougaar
This class is used to map the aggregation types and their corresponding values. This is used to create instances of aggregation types, which are listed to the user in the Query SnapIn. The class AMT.java uses this class to create objects of aggregation type.

public class ErrorLog extends JPanel implements ListSelectionListener
Graphical Component for displaying error information.

4.3.2 Seccon Communication Package Classes




MnRManagerList
#list: Vector = null
«create» +MnRManagerList()
«create» +MnRManagerList()
+toXML()
+create()
+get()
+add()
+size()
SCChannelAlert
#delay: int
«create» +SCChannelAlert()
«create» +SCChannelAlert()
+create()
+toXML()
+getConsoleManager()
+getDelay()
StreamProcessing
+formFeed: char = '\f'
«create» +StreamProcessing()
+get()
+generate()
+getAnswer()
+clean()
+receive()
+send_request()
MnRManagerInfo
«create» +MnRManagerInfo()
«create» +MnRManagerInfo()
+create()
+toXML()
+getmnrId()
+getcommunity()
SCManagerList
#list: Vector = null
«create» +SCManagerList()
«create» +SCManagerList()
+toXML()
+create()
+get()
+add()
+size()
SCReturnCode
«create» +SCReturnCode()
«create» +SCReturnCode()
+create()
+toXML()
+getQueryId()
+getCommand()
+getReturnCode()


SCManagerInfo
«create» +SCManagerInfo()
«create» +SCManagerInfo()
+create()
+toXML()
+getcmId()
+getaddress()
SCFactory
+TAG_Index()
+valid()
+new_instance()
+valid()
+create()
+create()
+create()
+create()
SCLayer
«create» +SCLayer()
+send()
+setUpperLayer()
+setLowerLayer()
+receive()
+wrap()
+unwrap()
SCMessage
«create» +SCMessage()
«create» +SCMessage()
+create()
+toXML()
+main()
DeleteQuery
«create» +DeleteQuery()
«create» +DeleteQuery()
+create()
+toXML()
+getID()

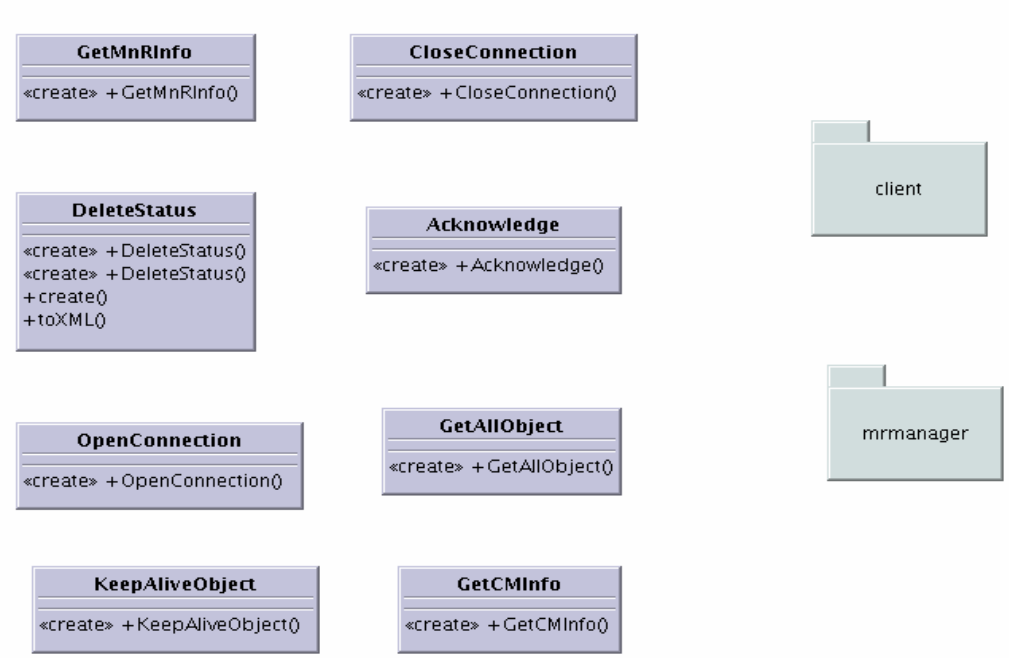


Figure 47: Classes in edu.memphis.issrl.seccon.communication package

public class Acknowledge extends SCObject
This object is sent by one component (SC client or SC manager) to the other in order to confirm a message's arrival.

public class AnalyzerResults extends SCObject
Stores the results produced by one analyzer (sensor or m&r)

public class CloseConnection extends SCObject
A close connection object (sent by a SC Client to the SC manager to close the alive connection previously established).

public class DeleteQuery extends SCObject
A delete query command sent by a SC client to a SC manager.

public class DeleteStatus extends SCObject
Abstract class of objects processed by the Security Console.

public class GetAllObject extends SCObject
Get all results command. Sent by a SC client to a SC manager. This command is used for crash recovering.

public class GetCMInfo extends SCObject
Get SC managers info from the society. Sent by a SC client to the Society SC manager.

public class GetMnRInfo extends SCObject
Get M&R managers info from the society. Sent by a SC client to the Society SC manager.

public class KeepAliveObject extends SCObject
I-AM-ALIVE message for checking the status of the connection.

public class MnRManagerInfo extends SCObject
Information of a M&R manager.

public class MnRManagerList extends SCObject
List of MNRManagerInfo object. (Information of the entire M&R manager in the society).

public class OpenConnection extends SCObject
Open connection command. Sent by a SC client to a SC manager.

public class PublishDetailsQuery extends SCObject
A publish details of a given query. Sent by a SC client to a SC manager.

public class PublishQuery extends SCObject
A publish query command. Sent by a SC client to a SC manager.

public class QueryResult extends SCObject
A query result us the collection of IDMEF Messages that satisfies the query associated with the stored id.

public class QueryResultList extends SCObject
A list of QueryResult object generated by the Security components.

public class SCChannelAlert extends SCObject
Message generated by the client when the communication channel is failing.

final public class SCFactory
A class with all the possible tags in the communication process

public class SCLayer implements Serializable, NotPersistable
Abstract class that represents a layer in a communication protocol for the Security Console.

public class SCManagerInfo extends SCObject
Information of a Security Console manager (SC manager).

public class SCManagerList extends SCObject
A list with the information of each SC manager in the society.

public class SCMessage extends SCObject
Communication message object between a SC client and a SC manager.

public class SCReturnCode extends SCObject
Information of the status of a command sent by the SC client. This information is generated by the SC manager and send back to the SC client.

public class SCTracker
Container for SC components. Similar to a simple hash table.

public class StreamProcessing
First layer in the communication protocol. This class is responsible for maintaining the physical connection and processing the character streams.

4.3.3 Seccon Communication Client Package Classes

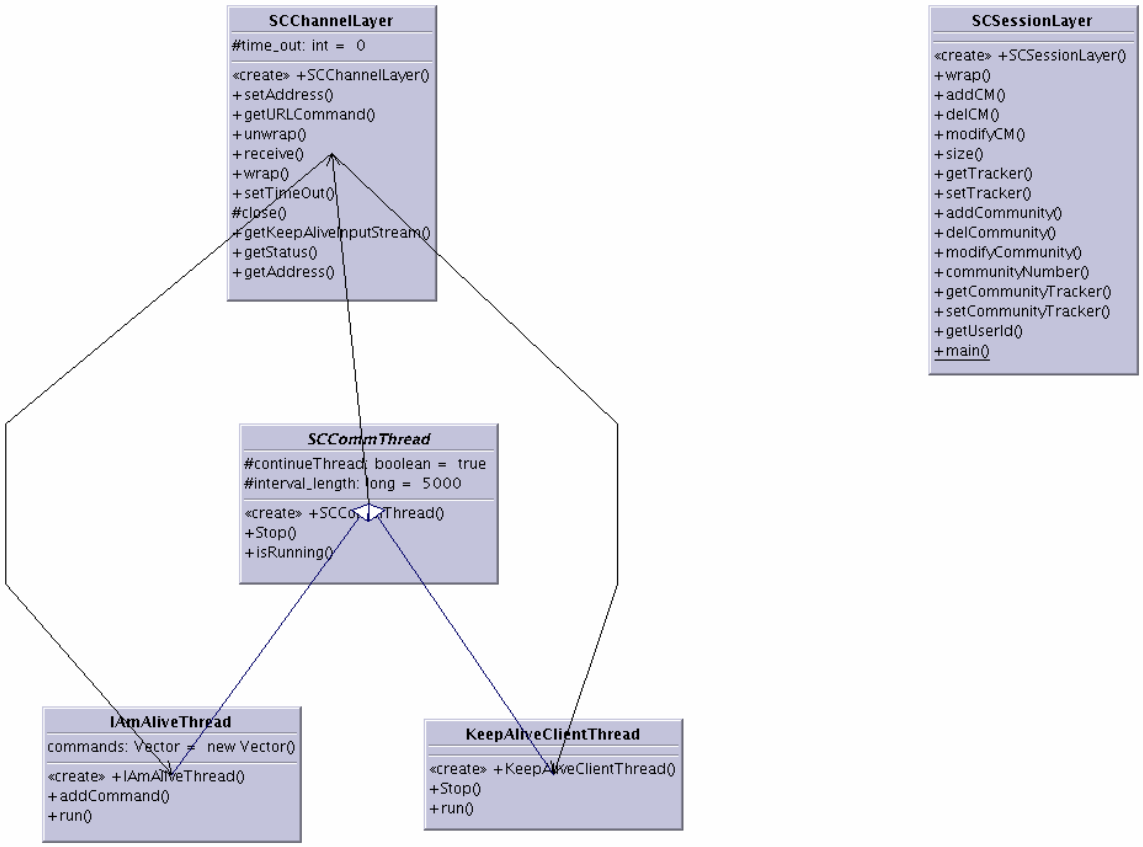


Figure 48: Classes in edu.memphis.issrl.seccon.communication.client package

public class SCChannelLayer extends SCLayer
Second layer in the communication protocol (client side). This layer is responsible of keeping the console manager information of the physical communication channel, the receiving thread.

public abstract class SCCommThread extends Thread
Dedicated thread to the physical connection

public class SCSessionLayer extends SCLayer
Third layer in the communication protocol (client side). This class is responsible of maintaining the different channel connections

public class IAmAliveThread extends SCCommThread
This deamon is permanently looking for the status of the connection

public class KeepAliveClientThread extends SCCommThread
Class for receiving the results from the M&R Manager using a keep alive connection

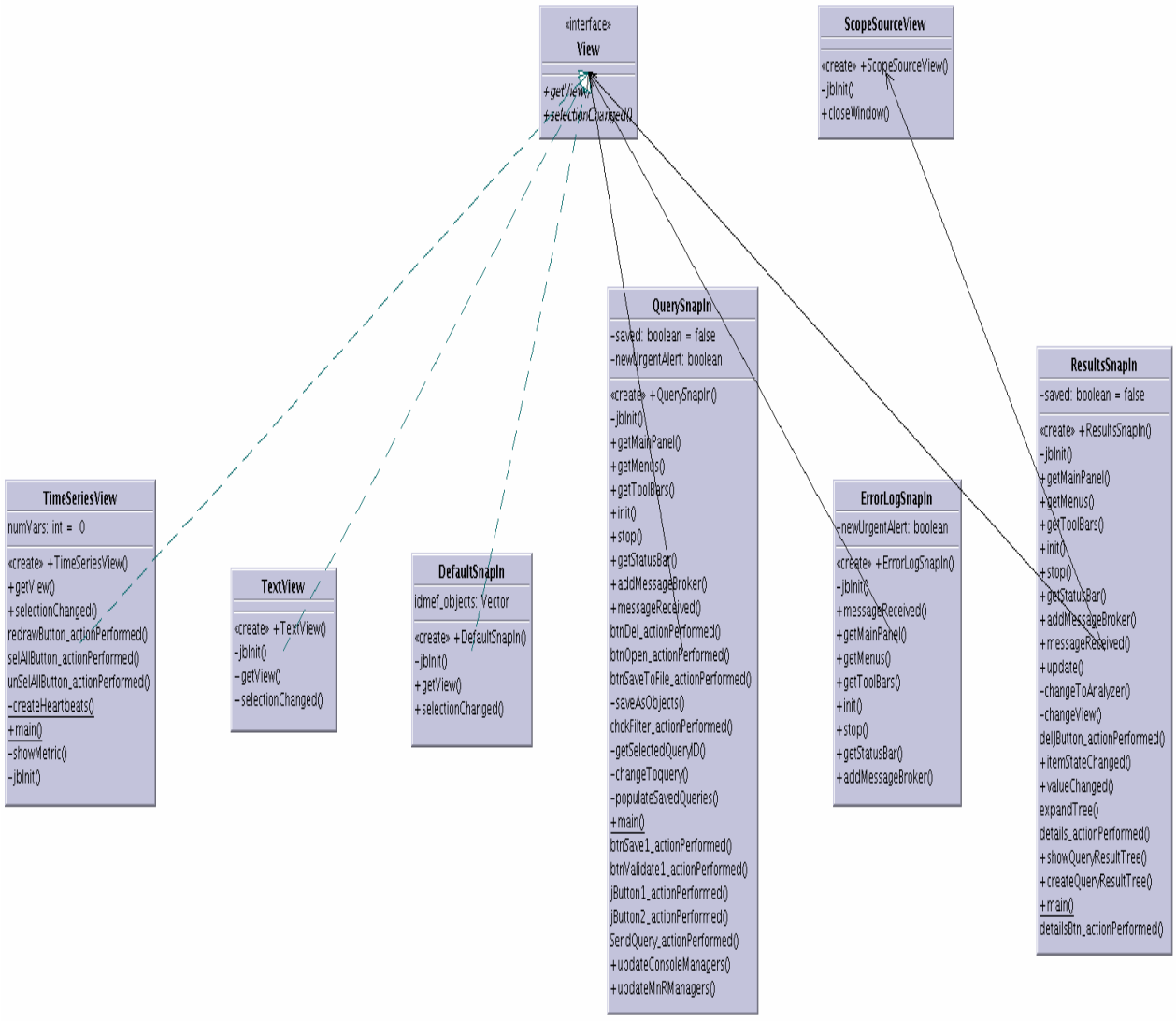
4.3.4 Seccon Communication MRManager Package Classes

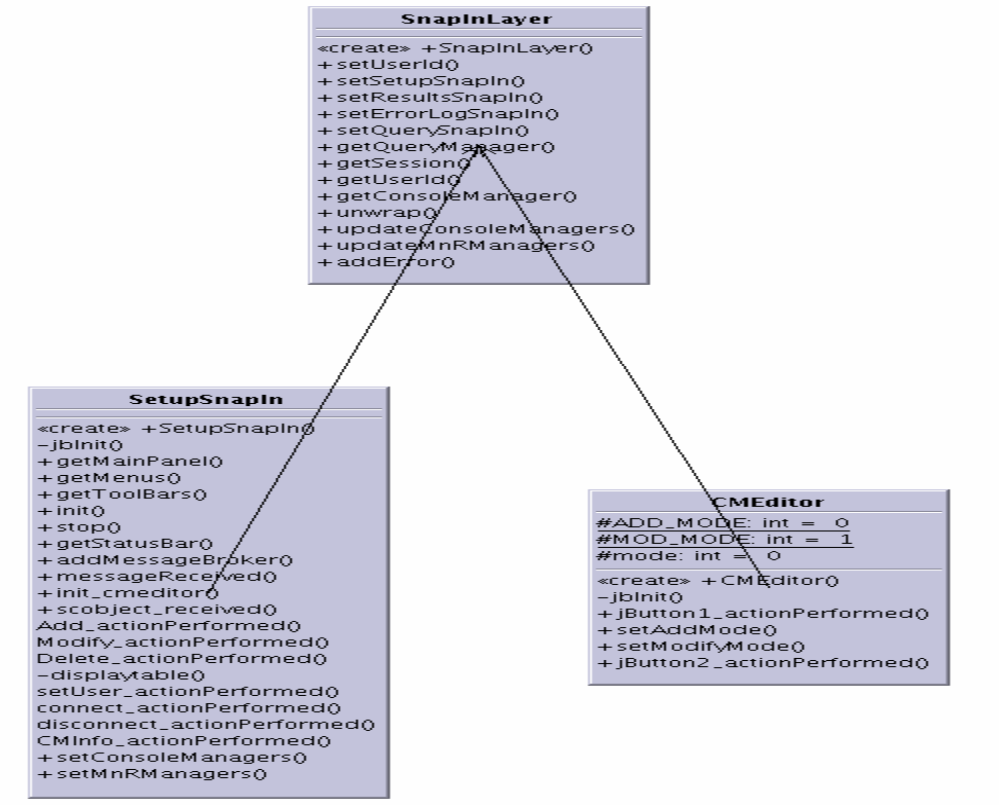
Figure 49: Classes in package edu.memphis.issrl.seccon.communication.mrmanager package

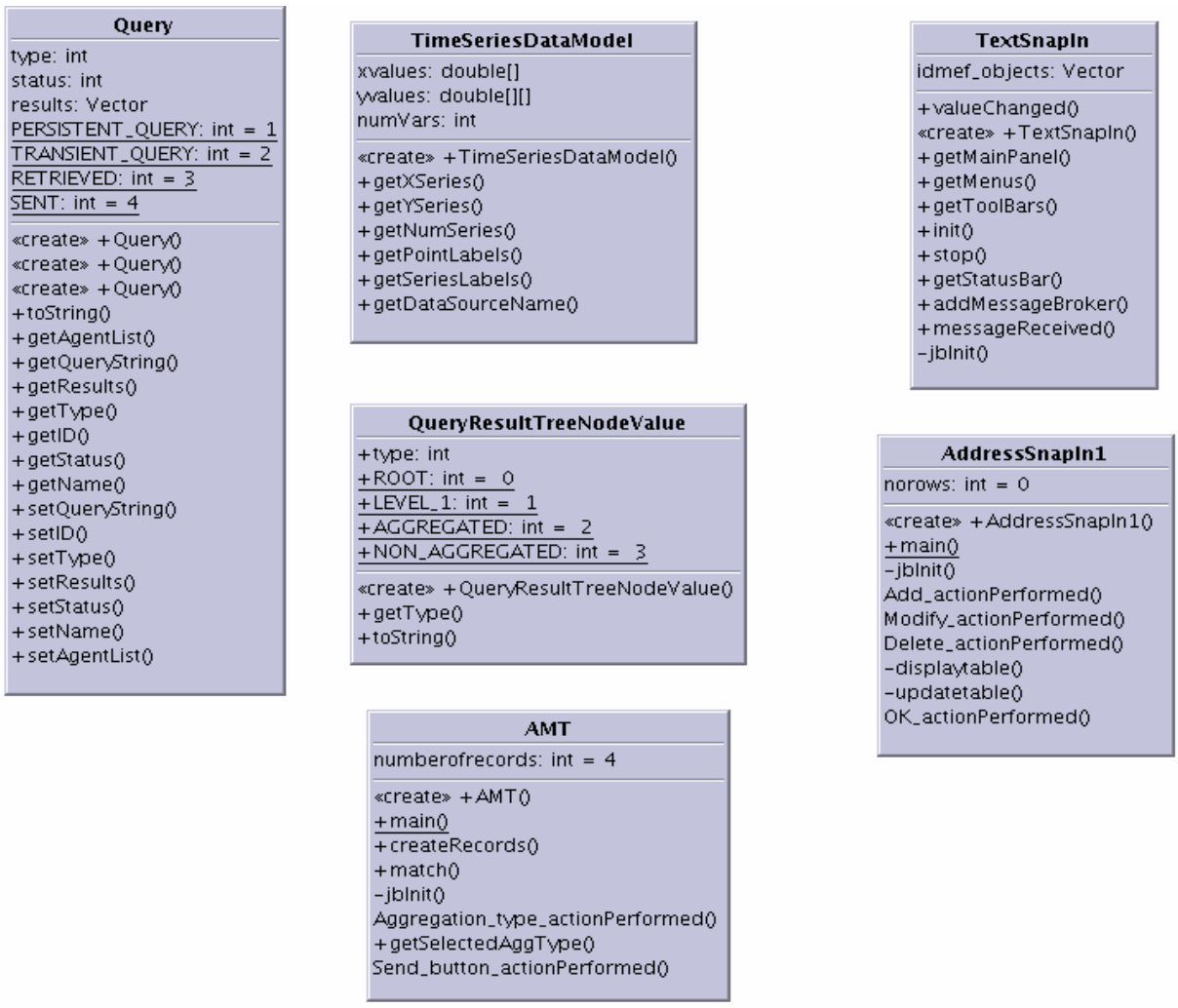
public class SCChannelLayer extends SCLayer
Second layer in the communication protocol (SC manager side). Responsible of tracing all the open servlets and report the command to the agent blackboard.

public class SCServletComponent extends BlackboardServletComponent
This keep-alive component is used for monitoring the aggregation agent's blackboard by way of incremental updates passed over a keep alive connection.

4.3.5 Seccon GUI Package Classes




SnapInLayer
«create» +SnapInLayer()
+setUserId()
+setSetupSnapIn()
+setResultsSnapIn()
+setErrorLogSnapIn()
+setQuerySnapIn()
+getQueryManager()
+getSession()
+getUserId()
+getConsoleManager()
+unwrap()
+updateConsoleManagers()
+updateMnRManagers()
+addError()
SetupSnapIn
«create» +SetupSnapIn()
-jbInit()
+getMainPanel()
+getMenus()
+getToolBars()
+init()
+stop()
+getStatusBar()
+addMessageBroker()
+messageReceived()
+init_cmeditor()
+scobject_received()
Add_actionPerformed()
Modify_actionPerformed()
Delete_actionPerformed()
-displaytable()
setUser_actionPerformed()
connect_actionPerformed()
disconnect_actionPerformed()
CMInfo_actionPerformed()
+setConsoleManagers()
+setMnRManagers()
CMEditor
#ADD_MODE: int = 0
#MOD_MODE: int = 1
#mode: int = 0
«create» +CMEditor()
-jbInit()
+jButton1_actionPerformed()
+setAddMode()
+setModifyMode()
+jButton2_actionPerformed()


Query
type: int
status: int
results: Vector
PERSISTENT_QUERY: int = 1
TRANSIENT_QUERY: int = 2
RETRIEVED: int = 3
SENT: int = 4
«create» +Query()
«create» +Query()
«create» +Query()
+toString()
+getAgentList()
+getQueryString()
+getResults()
+getType()
+getID()
+getStatus()
+getName()
+setQueryString()
+setID()
+setType()
+setResults()
+setStatus()
+setName()
+setAgentList()
TimeSeriesDataModel
xvalues: double[]
yvalues: double[][]
numVars: int
«create» +TimeSeriesDataModel()
+getXSeries()
+getYSeries()
+getNumSeries()
+getPointLabels()
+getSeriesLabels()
+getDataSourceName()
TextSnapIn
idmef_objects: Vector
+valueChanged()
«create» +TextSnapIn()
+getMainPanel()
+getMenus()
+getToolBars()
+init()
+stop()
+getStatusBar()
+addMessageBroker()
+messageReceived()
-jbInit()
QueryResultTreeNodeValue
+type: int
+ROOT: int = 0
+LEVEL_1: int = 1
+AGGREGATED: int = 2
+NON_AGGREGATED: int = 3
«create» +QueryResultTreeNodeValue()
+getType()
+toString()
AddressSnapIn1
norows: int = 0
«create» +AddressSnapIn1()
+main()
-jbInit()
Add_actionPerformed()
Modify_actionPerformed()
Delete_actionPerformed()
-displaytable()
-updatetable()
OK_actionPerformed()
AMT
numberofrecords: int = 4
«create» +AMT()
+main()
+createRecords()
+match()
-jbInit()
Aggregation_type_actionPerformed()
+getSelectedAggType()
Send_button_actionPerformed()

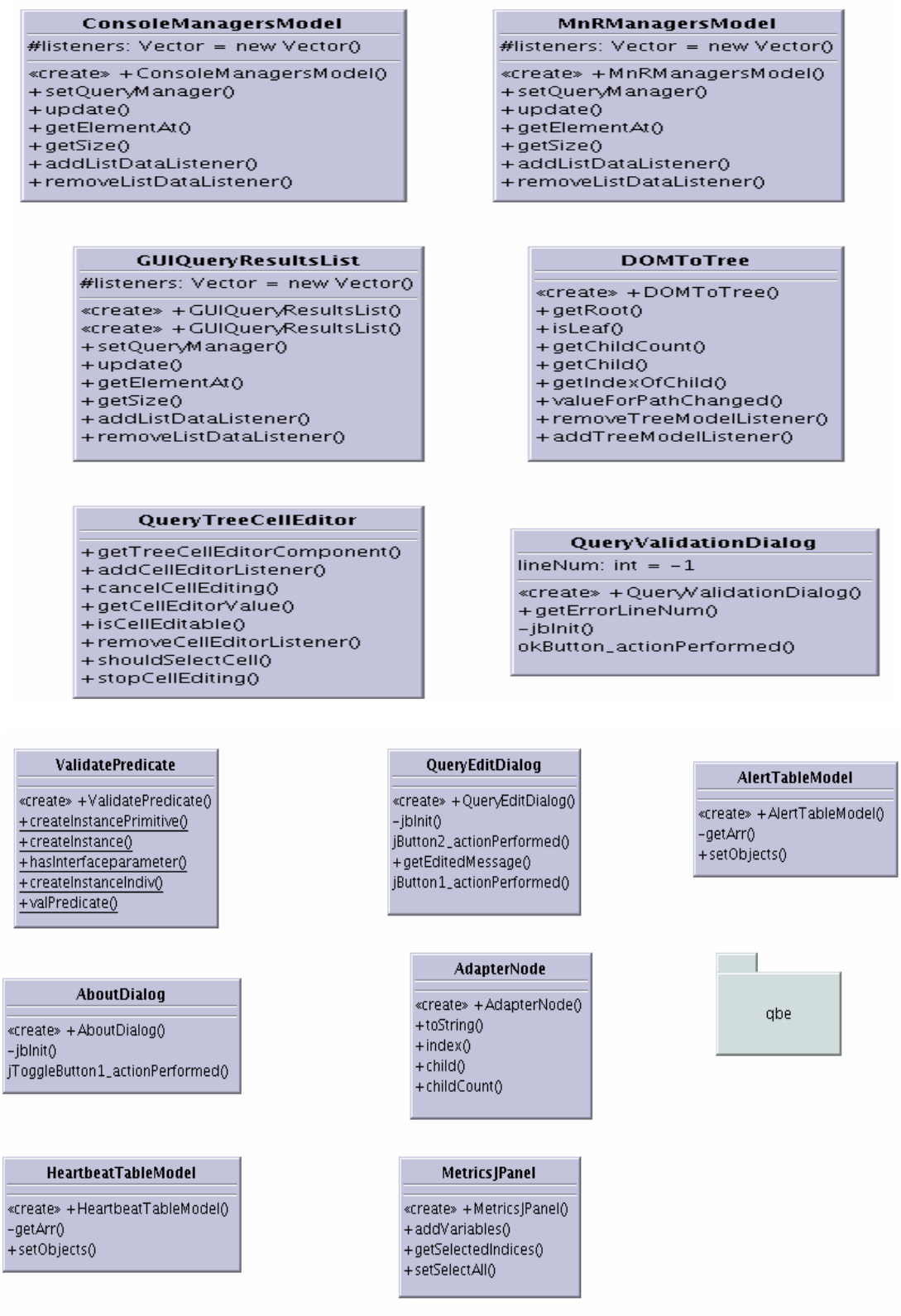


Figure 50: Classes in edu.memphis.issrl.seccon.gui package

public class AboutDialog extends JFrame
This frame that opens on pressing menu Help → About. This shows the Version of the product and the team working on it.

public class AlertTableModel extends DefaultTableModel
This class is table model for Alert Objects

public class AMT extends JPanel
Used to maintain a list of aggregation method types and their corresponding parameters and values. This creates aggregation type objects using AggType_Cougaar.java class. This list is provided to the user by the QuerySnapIn.java class

public class CMEditor extends JDialog

public class ConsoleManagersModel implements ListModel
The list model for the Security Console Managers Information. This model is used in the GUI List and combo box of Security Console Managers

public class DefaultSnapIn extends JPanel implements View
Default snapin provides tree view to idmef objects.

public class DOMToTree implements TreeModel
Converts xml to tree form and serves as a tree model

public class ErrorLogSnapIn extends JPanel implements SnapIn
Results SnapIn takes in query from user and sends it to M&R manager thru QueryManager. It can apply query over query. Queries can be saved to file and can be retrieved. It also shows the results of retrieved queries. They can be sorted and can be viewed in different views. Views are pluggable in to this snapin. For a class to become a view for IDMEF objects it should implements 'View' interface.

public class GUIQueryResultsList implements ListModel
The tree model for the Query Results obtained from the Security Console Managers. This model is used in the GUI Tree that shows the results

public class HeartbeatTableModel extends DefaultTableModel
Table model for heart beat objects

public class MetricsJPanel extends JPanel
This panel presents the available metrics and allows the user to choose which metrics to draw

public class MnRManagersModel implements ListModel
The list model for the M&R Managers Information. This model is used in the GUI List and combo box of M&R Managers

public class Query implements Serializable
Query represents Standard Query that is sent to M&R Manager and Filter that is applied on Standard Query Results.

public class QueryEditDialog extends JDialog
Dialogbox for editing the content of a Query

public class QueryResultTreeNodeValue
This is used to create tree nodes with the given name and the type. The type can be the root element, the first level (which represents the non expanded queries), the aggregated queries which can further be expanded and non-aggregated queries which cannot be expanded further. This class is used by the ResultsSnapIn class to determine the result type. The result can be an aggregated query which can be further expanded or non aggregated query which cannot be expanded.

public class QuerySnapIn extends JPanel implements SnapIn
Results SnapIn takes in query from user and sends it to M&R manager thru QueryManager. It can apply query over query. Queries can be saved to file and can be retrieved. It also shows the results of retrieved queries. They can be sorted and can be viewed in different views. Views are pluggable in to this snapin. For a class to become a view for IDMEF objects it should implements 'View' interface.

public class QueryValidationDialog extends JDialog
This dialog shows the result of validating a Jython predicate.

public class ResultsSnapIn extends JPanel implements SnapIn, ItemListener,ListSelectionListener
Results SnapIn takes in query from user and sends it to M&R manager thru QueryManager. It can apply query over query. Queries can be saved to file and can be retrieved. It also shows the results of retrieved queries. they can be sorted and can be viewed in different views. Views are pluggable in to this snapin. For a class to become a view for IDMEF objects it should implements 'View' interface

public class ScopeSourceView extends JFrame
Setup snapin.Displays GUI for setting up host address and capabilities object.

public class SetupSnapIn extends JFrame implements SnapIn,SCObjectListener
Setup snapin.Displays GUI for setting up host address and capabilities object.

public class SnapInLayer extends SCLayer
Last layer in the communication protocol (client side). This layer is responsible of receiving information from the protocol and sending to the corresponding graphical component. Also, of receiving commands from the graphical components and sending to the security console manager through the communication protocol

public class TextView extends JPanel implements View
Text View of IDMEF objects

public class TimeSeriesDataModel implements LabelledChartDataModel, ChartDataModel
This component allows the user to display the results in Time Series format.

public class TimeSeriesView extends JPanel implements View

Graphical user interface to show to the user metrics information. It gets the information from PSP_Monitor in XML format and parse it to fill the information in a table. The information is shown too in a graphical way using a JClass Chart component

public class ValidatePredicate
The purpose of this class is to validate a Jython predicate. It performs syntactical and some semantic validations.

public interface View
This interface should be implemented by classes, which want to be plugged in to views list.

public class CleanNode

CleanNode is a simple Utility class that takes in any Root Node not the Document Node but it's child and iteratively searches for the trash text Nodes. It successively gets rid of any of these nodes by reaching its parent Node and calling the DOM API's *removeChild* (*org.w3c.dom.node*) method. It has on static method for cleaning trash nodes.

public class EpisodeTree extends JPanel implements TreeSelectionListener

GUITree is a custom class that uses the simple and basic DefaultmutableTreeNode class to create tree nodes. It handles Single Tree Node Selection and takes actions. It also has tool tips and rendering of selective nodes.

public class GUITable extends JPanel

GUITable is a custom class that incorporates a custom table Model inserting a model -- a sorter -- between the table and its data model. It also has column tool tips and rendering of selective cells.

public class FileChooserFilter extends FileFilter

The Custom Class for filtering files of type .xml & .log only.

public class Utils

Utils defines the file extension types allowable for selection.

public class TableSorter extends AbstractTableModel

TableSorter is a decorator for TableModels; adding sorting functionality to a supplied TableModel. TableSorter does not store or copy the data in its TableModel; instead it maintains a map from the row indexes of the view to the row indexes of the model. As requests are made of the sorter (like getValueAt(row, col)) they are passed to the underlying model after the row numbers have been translated via the internal mapping array. This way, the TableSorter appears to hold another copy of the table with the rows in a different order.

4.3.6 Seccon GUI Chart Package Classes

The classes in this package use the third party library software, ***jfreechart***, ***version 0.9.19***. The Bar Charts and the Time Charts in this library have been utilized exhaustively for depicting all

the graphical statistics necessary for the Visualization. These visuals have the ability to zoom, print and change chart visual properties.

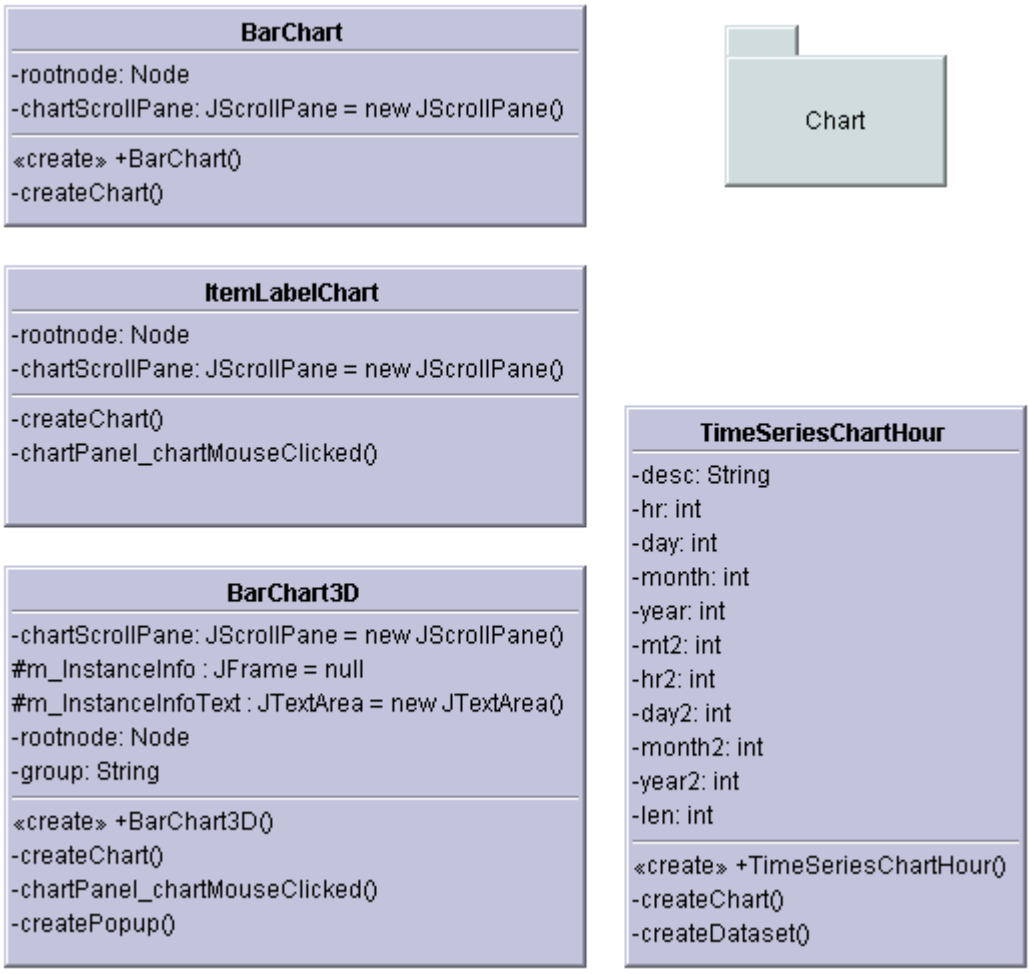


Figure 51: Classes in edu.memphis.issrl.seccon.gui.chart package

public class BarChart extends javax.swing.JPanel

The 2-D bar chart for showing the summarized distribution of selected Messages with respect to the Date & Time of detection and their numbers. There is provision to analyze the displayed chart (like Zoom, change properties, print copy, save) **.**

public class ItemLabelChart extends JPanel

Item Label Chart Class showing a label generator that displays labels that include a percentage calculation. It incorporates a class that is a custom label generator. The distribution of Episodes with respect to its numbers and percentage is displayed. There is provision to analyze the displayed chart (like Zoom, change properties, print copy, save).

public class BarChart3D extends javax.swing.JPanel

Bar chart showing the category information, Count of Messages in each category type. Also responds to mouse selection for further getting the details as a time series. There is provision to analyze the displayed chart (like Zoom, change properties, print copy, save).

public TimeSeriesChartHour extends javax.swing.JPanel

TimeSeriesChartHour chart is used for showing the time sequencing of the activity of either the Security Analyzer or Source of Attack or the Target of Attack. There is provision to analyze the displayed contents on the chart and chart itself (like Zoom, change properties, print copy, save).

4.3.7 Seccon GUI Graph Package Classes

The classes in this package use the third party library software, ***jgraph, version 5.0 and jgraphaddons, version 1.0***. The MultipleLine Cell class incorporates the fundamental features of these libraries.

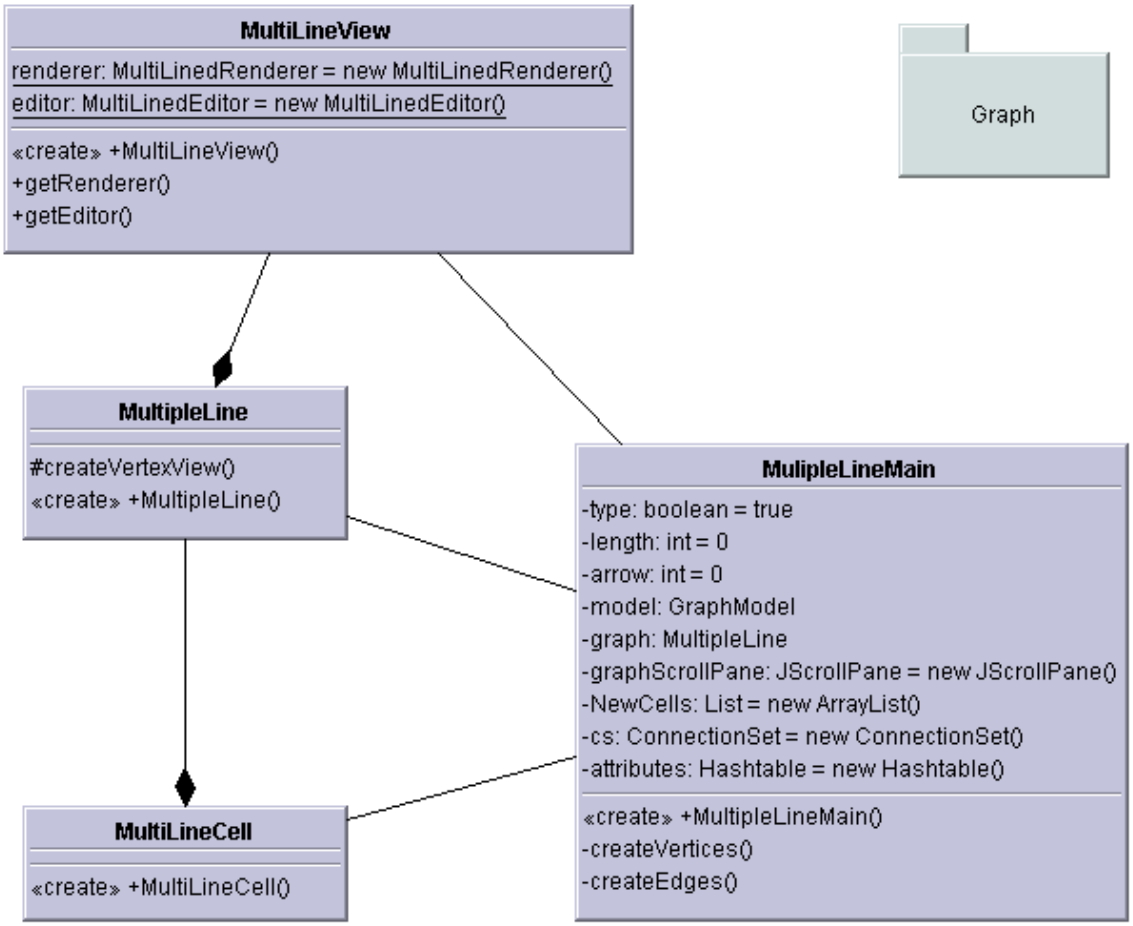


Figure 52: Classes in edu.memphis.issrl.seccon.gui.graph package

public class MultiLineCell extends DefaultGraphCell

This class inherits from DefaultGraphCell and provides the constructor for creating a MultiLineCell.

public class MultiLineView extends VertexView

This Class creates a Multiline Renderer that provides a view for creating a MultipleLineView of VertexView.

public class MultipleLine extends JGraph

This class creates a custom View by overriding JGraph's *createVertexView* method. Here a MultiLineCell is created.

public class MultipleLineMain extends javax.swing.JPanel

This main Class that creates and populates the individual *MultilineCell*. Multiple Lines for each Editable Cell. Uses the *GraphModel*

4.3.8 Seccon GUI Xmining Package Classes

public class DomToTreeModelAdapter implements javax.swing.tree.TreeModel

This adapter converts the current Document (a DOM) into a JTree model. It can respond with an Adapter Object to a tree selection.

public class AdapterNode

It is a custom class having defined methods to traverse the DOM.

public class XmineSnapIn extends JPanel implements Snap In

The main class for the Security Console XML messages mining. Lets the user create, open, save, configure, datasets, and perform analysis. It is the parent class for initializing the database driver and initializing the other subclass modules – *DataPreProcess*, *StatisticalAnalysis*, *FrequentEpisodeMain* and *AssociationRules.Visual.ARViewer*.

public class StatisticalAnalysis extends JPanel implements TreeSelectionListener

This panel provides the user interface to the Summary information on Alerts and MnR messages. Uses Graphical Tools (Charts) to provide Visualization to the summarized Results.

public class FrequentEpisodeMain extends JPanel

This panel is the main Mining Panel that inputs and outputs the parameters and the Mining results respectively. For Visualization, the JTree, Graph and Chart are used.

public class DataPreProcess extends JPanel

This Class mainly handles the DataBase management for the System Administrator by providing facilities to Alter, Upload, Delete and Select the records for further transaction. It has methods to interface with database and process records.

4.3.9 Seccon GUI Xmining Query Package Classes

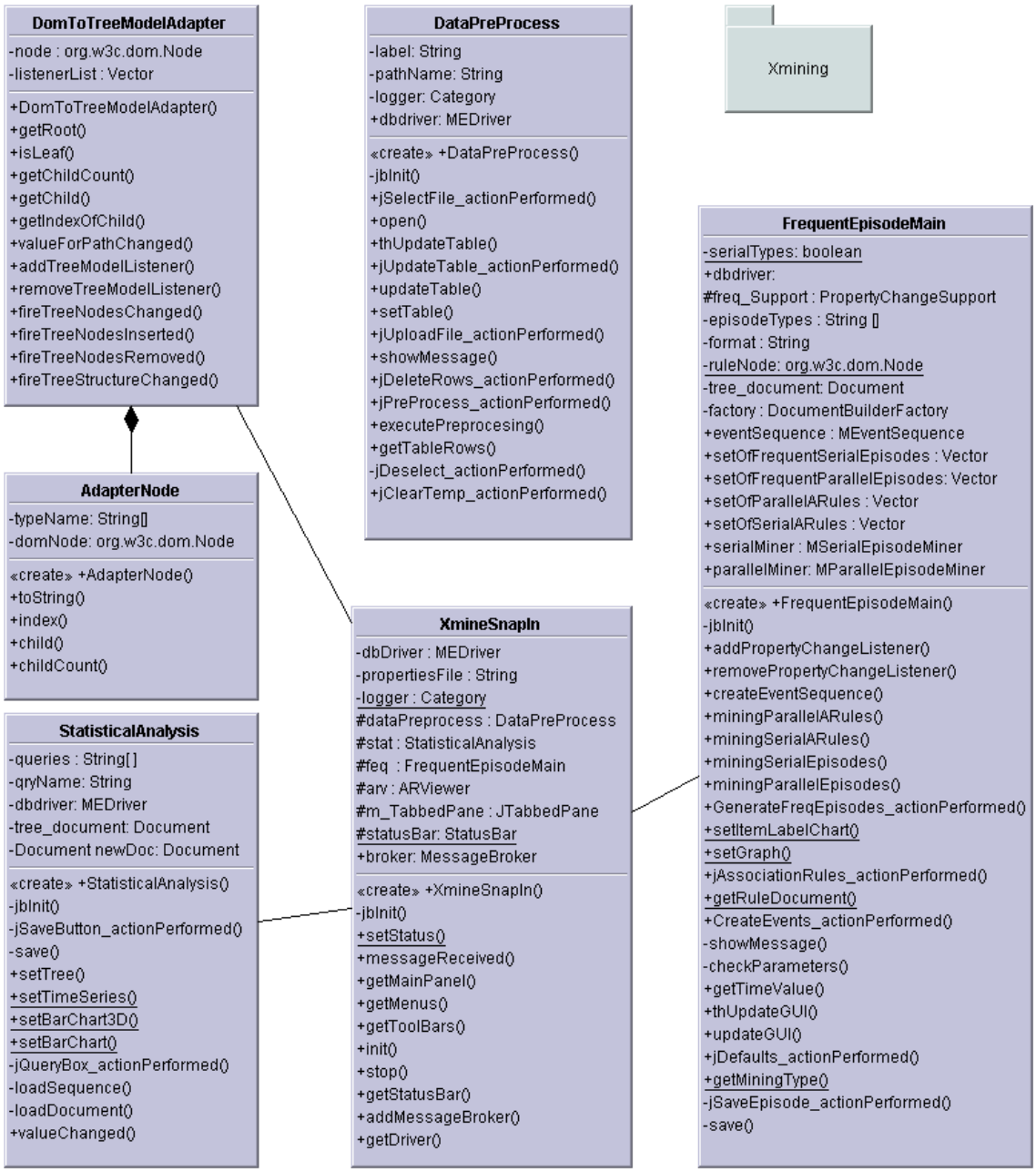


Figure 53: Classes in edu.memphis.issrl.seccon.gui.Xmining package

public class DatabaseQuery

public class SequenceQuery implements Runnable

This class creates the Chart Data for showing the statistics of Message density over a period of time for the range of messages that the User has interest.

public class ManagerQuery implements Runnable

This class creates the Chart Data for showing the statistics of Analyzer Messages over a period of time for the range of messages that the User has interest.

public class VictimListQuery implements Runnable

This class processes the xml query result exclusively for Victims. It provides the basic dataset containing messages - victim_messages, quantity - Count, timing - TimeStamp.

public class VictimTimeQuery

This class creates a Time Series Data for the Particular Target/Attacker Element. It responds to the message sent to it by the *VictimListQuery* class. It processes the Document containing Attacker/ Target Data for it's time distribution. Uses class *TimeSeriesChartHour*.

public class MessageQuery

The class responding to the message sent to it by the *ManagerQuery* class. Represents the distribution of each Analyzer according to time. Uses class *TimeSeriesChartHour*.

public class GraphEventsQuery

GraphEventsQuery populates the data for *MultiLineCell* with the events for each selected Episode (whether Serial/ Parallel).

4.3.10 Seccon GUI AssociationRules Package Classes

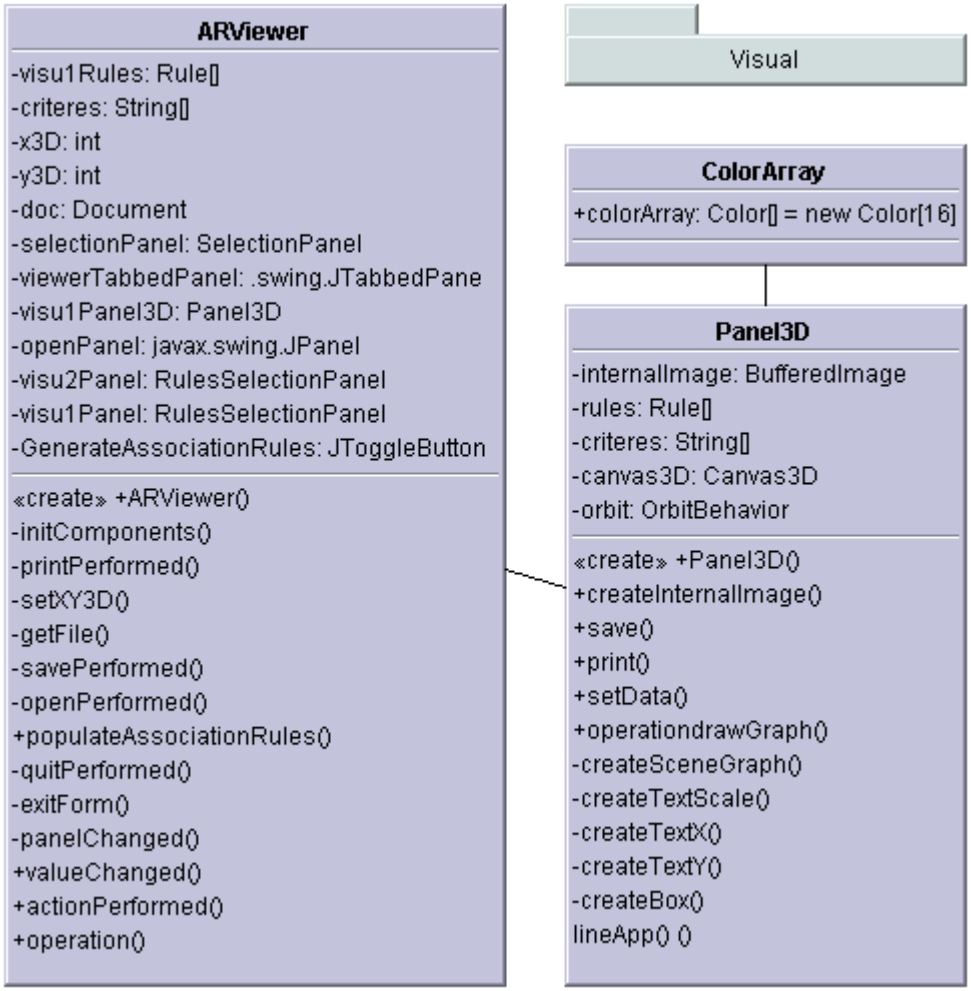


Figure 54: Classes in edu.memphis.issrl.seccon.gui.AssociationRules.Visual package

public class ARViewer extends javax.swing.JPanel

Association Rules Viewer is the main Class that controls the viewing of rules, rearranging the order of antecedents and consequents according to the sorting criteria that involves the use of custom classes in *sortedListPanel*. It contains the Panel to switch to a graphical view of the association between the left and right sides of the rules, namely antecedent and consequent. Controls the other class *Panel3D*.

public class ColorArray

ColorArray class defines the set of Colors Available to cycle through, to paint the Visualization Objects. This class can be modified as per user needs.

public class Panel3D extends DataPanel implements Printable

Panel3D creates a Unique Visualization in 3-Dimensional using the supplied Rules and Criteria List. It implements Printable interface to save the Image or Print the Image visible in the view.

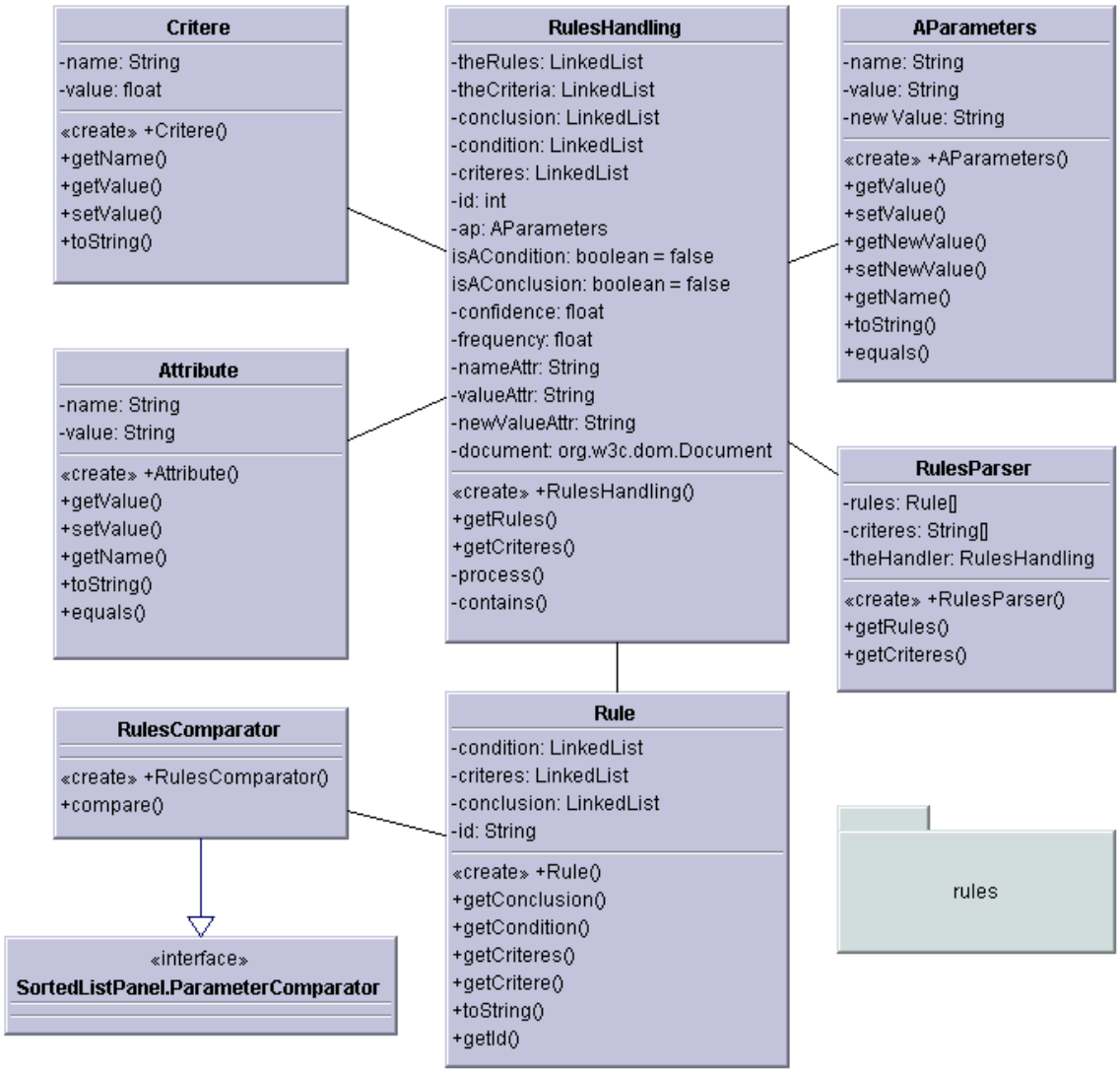


Figure 55: Classes in edu.memphis.issrl.seccon.gui.AssociationRules.rules package

public class AParameters

AParameters creates an Object with three parameters (Attributes)--classification, Source, Target.

public class Attribute

Attribute class creates an Object with two parameters (Attributes)--Confidence/Support and corresponding value for each Criteria tag in the generated Rule.

public class Critere

Critere creates an Object with two parameters (Attributes) -- Support/Confidence and its corresponding Value.

public class Rule

Rule class creates a typical object of LHS and RHS of Rules. It also provides an Object to create Object for Criteria. Multiple rules belonging to each of RHS and LHS is provided. This class is modified from Original.

public class RulesComparator extends ParameterComparator

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. Note: this comparator imposes orderings that are inconsistent with equals.

public class RulesHandling

RulesHandling Class is the Handler that parses the XML Document of rules.

public class RulesParser

This Class parses the given XML File. The Class returns two Collections - Rule and Criterias required by class Panel3D. It uses the class *RulesHandling*.

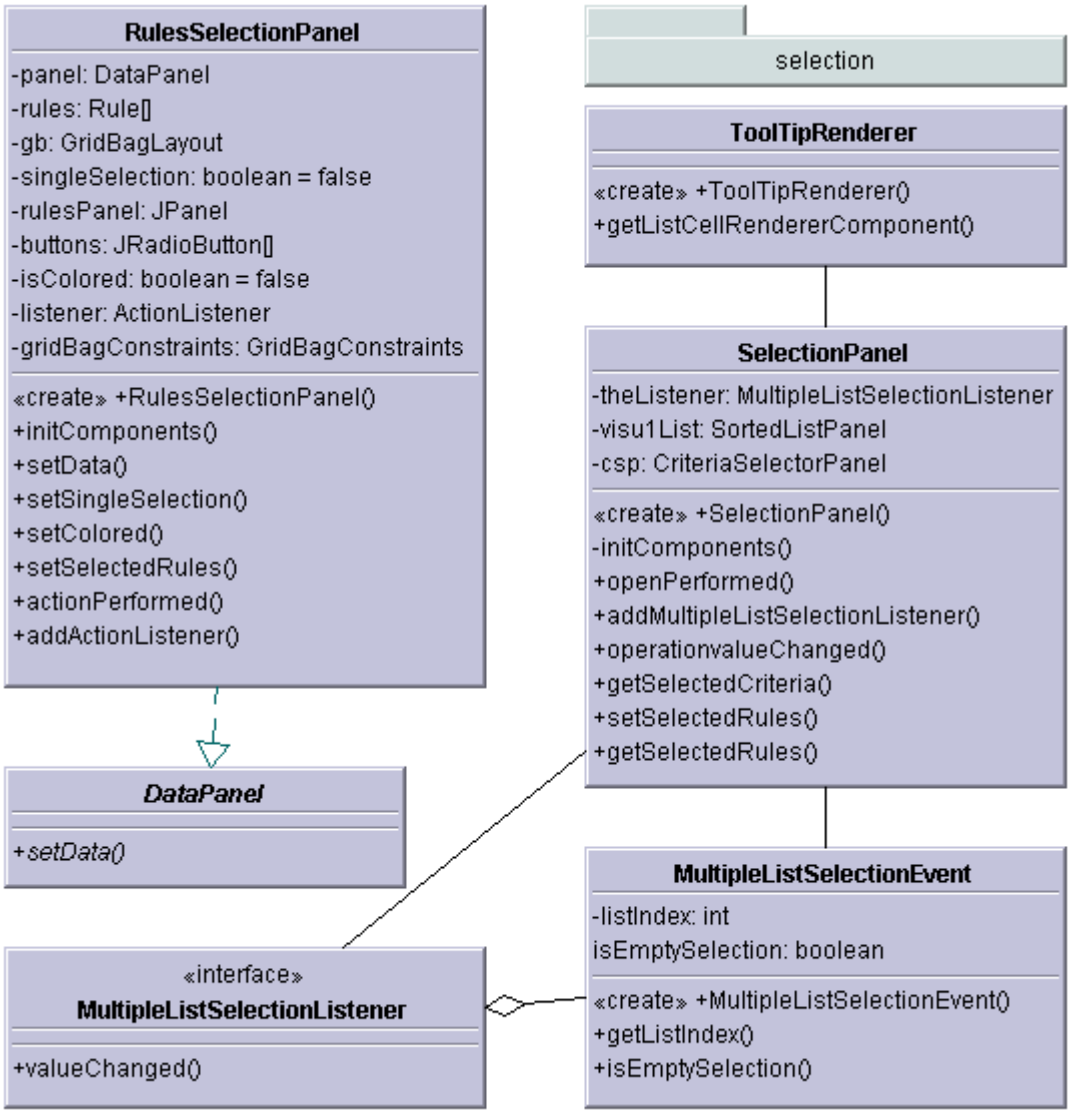


Figure 56: Classes in edu.memphis.issrl.seccon.gui.AssociationRules.selection package

public abstract class DataPanel extends JPanel

The Abstract Class that sets the Rules and Criteria.

public class SelectionPanel extends javax.swing.JPanel implements ListSelectionListener

SelectionPanel holds and controls the *Rules Panel* and *Criteria Selection* panel on GUI.

public class RulesSelectionPanel extends JPanel

This Class sets up the Rules Panel and populates it with the Rules. Basically sets up the GUI.

public interface MultipleListSelectionListener extends EventListener

MultipleListSelectionListener can be implemented for the List Selection Change Events.

public class MultipleListSelectionEvent extends ListSelectionEvent

MultipleListSelectionEvent is a Custom class for responding to the list Selection and Change actions.

public class ToolTipRenderer extends JLabel implements ListCellRenderer

ToolTipRenderer returns a component that has been configured to display the specified value. That component's paint method is then called to "render" the cell. If it is necessary to compute the dimensions of a list because the list cells do not have a fixed size, this method is called to generate a component on which *getPreferredSize* can be invoked.

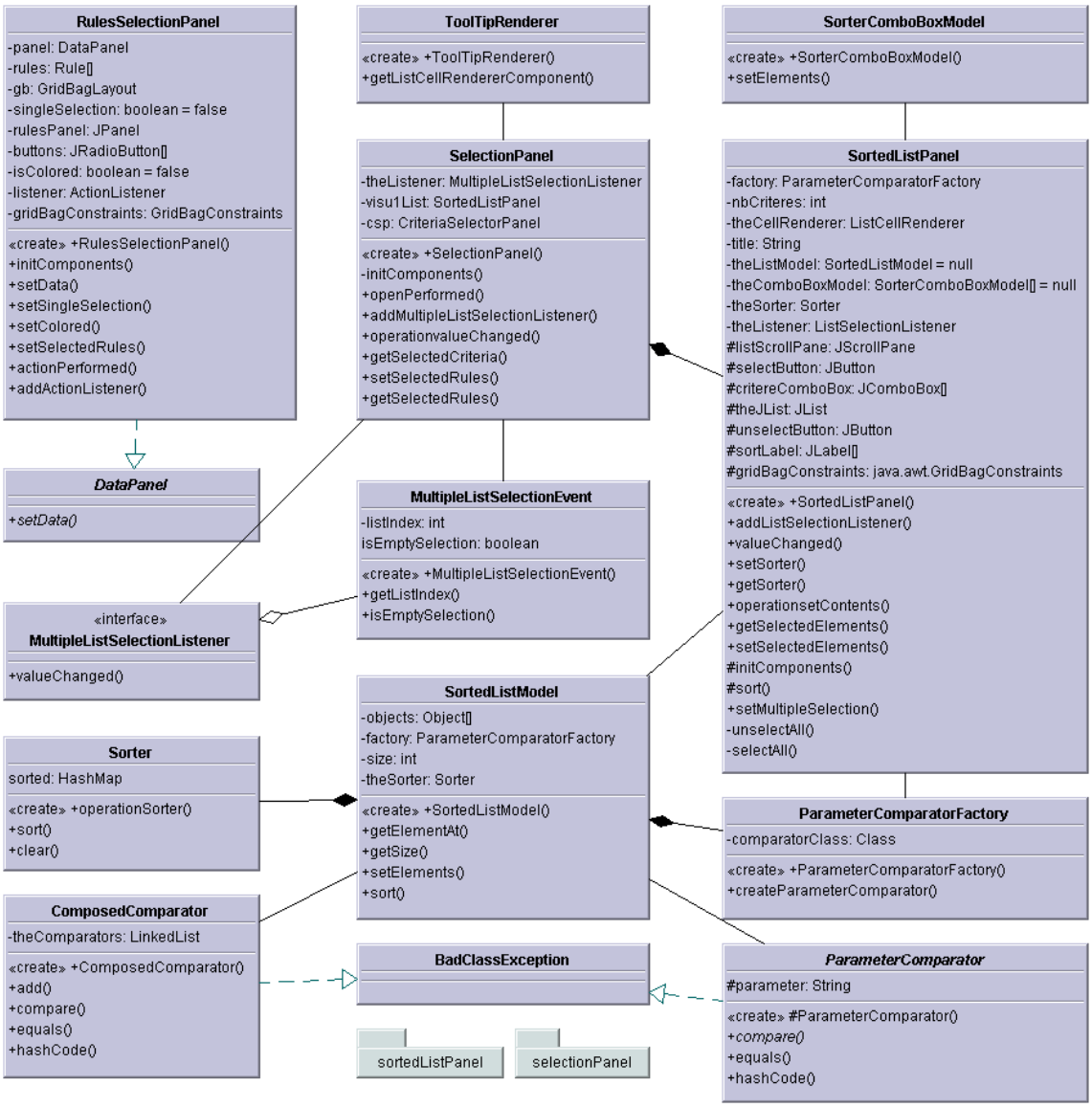


Figure 57: Classes in edu.memphis.issrl.seccon.gui.AssociationRules.ARViewer.sort package

public class SortedListPanel extends JPanel implements ListSelectionListener

Creates a SortedListPanel with a comparator- *AbstractParameterComparator* used to sort the list. nbCriteres - the number of ComboBoxes used to sort the list as well as a *ListCellRenderer* used to show the elements of the list.

public class SortedListModel extends javax.swing.AbstractListModel

SortedListModel ia a custom model for creating a *SortedListModel* .Uses the sorter class for sorting the Linked List of Criterias.

public class SorterComboBoxModel extends javax.swing.DefaultComboBoxModel

SorterComboBoxModel is a ComboBoxModel for Sorted List.

public class Sorter

Sorter class sorts a table of objects and storing in a HashMap.

public class ParameterComparatorFactory
ParameterComparatorFactory is a factory class for Comparators. Uses class *comparatorClass*.

public class ComposedComparator implements java.util.Comparator

Compares two object arguments which is required for ordering.

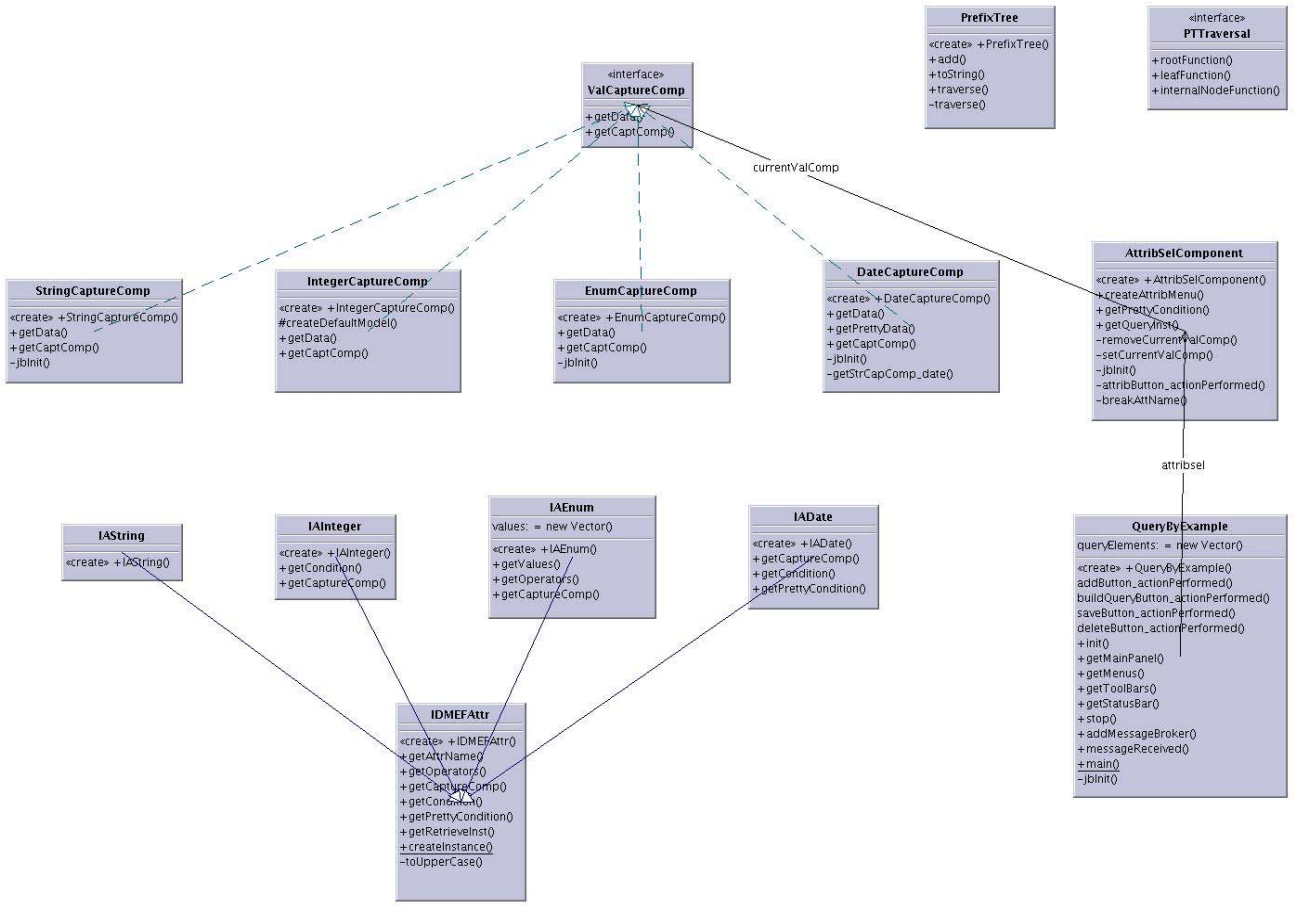


Figure 58: Classes in edu.memphis.issrl.seccon.qbe package

public class QueryByExample extends javax.swing.JFrame implements SnapIn

Query By Example helps user in building a query. It reads the information about the IDMEF attributes from a file. This information specifies the name and type of each attribute.

Interface ValCaptureComp

Interface representing the swing component needed to capture a value for an attribute.

public class AttribSelComponent extends javax.swing.JPanel

This component allows the user to choose an attribute of an IDMEF Alert

public class DateCaptureComp extends javax.swing.JPanel implements ValCaptureComp

Component to capture dates

public class EnumCaptureComp extends javax.swing.JPanel implements ValCaptureComp

Component to capture enumerated values

public class EnumCaptureComp extends javax.swing.JPanel implements ValCaptureComp

Component to capture enumerated values

public class StringCaptureComp extends javax.swing.JPanel implements ValCaptureComp

Component to capture string values

public class IADate extends IDMEFAttr

Class to represent Alerts of date type

public class IAEnum extends IDMEFAttr

Class to represent Alerts of enum type

public class IAInteger extends IDMEFAttr

Class to represent Alerts of numeric type

public class IAString extends IDMEFAttr

Class to represent Alerts of string type

public class IDMEFAttr extends java.lang.Object

Creates the appropriate Alert class

public class PrefixTree extends java.lang.Object

This class represents a prefix tree. It stores vectors of objects merging on unique branch vectors with common prefixes.

For instance, a prefix tree that stores the vectors: [1,2,3], [1,2,4], [1,5,6] has the following structure: (1 (2 (3 4)) (5 (6)))

4.3.11 Seccon MrManager Package Classes

public class AggQuery
Creates an AggQuery object with the basic information of the query

public class ChannelLayerPredicate implements UnaryPredicate
Unary Predicate to query the blackboard for the Channel Layer Object. Typically there's only one channel layer object.

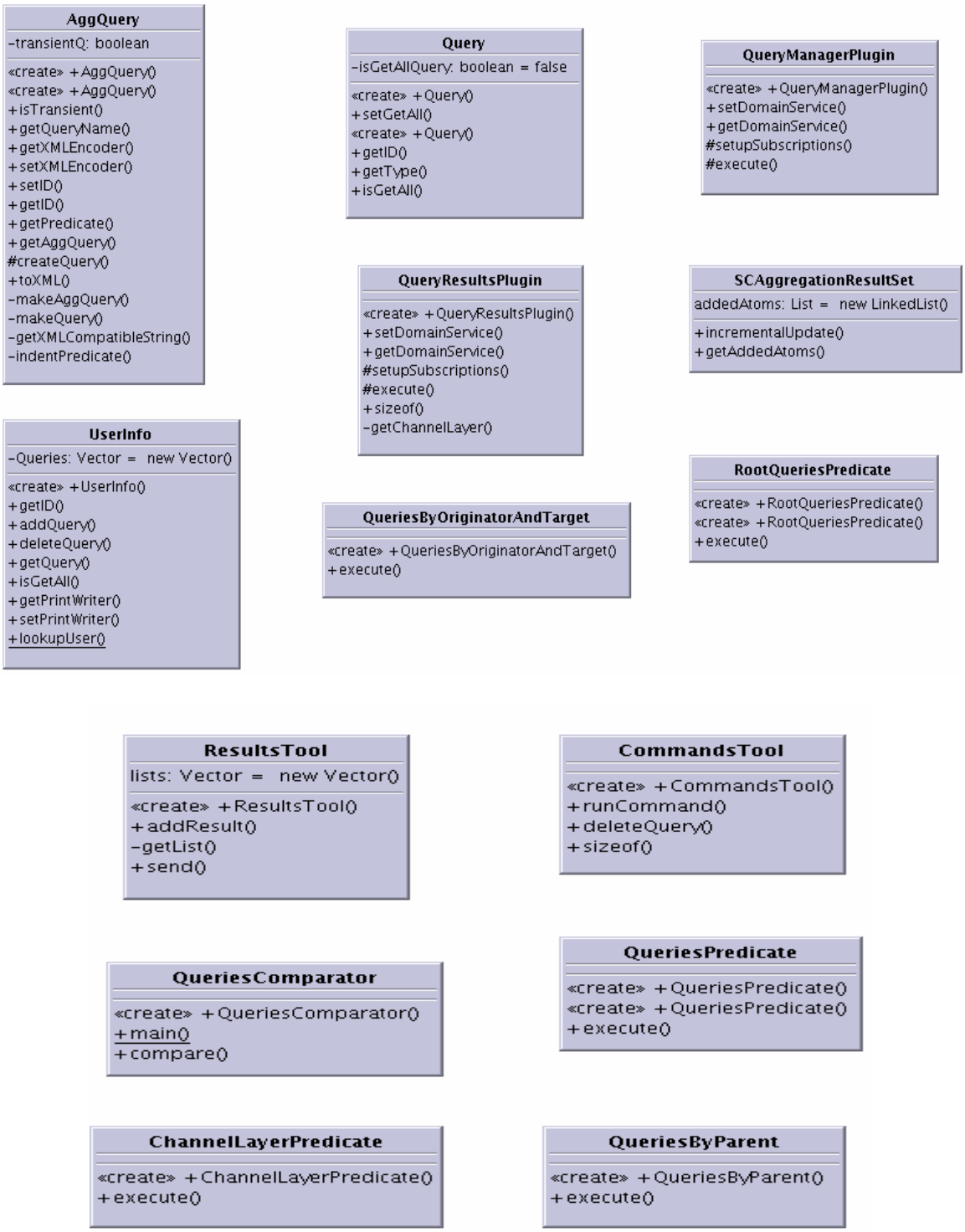


Figure 59: Classes in edu.memphis.issrl.seccon.mrmanager package

public class CommandsTool
The commands tool class is used by the Query Manager Plugin to processes the commands sent by the clients. The tool publishes queries and expansions, deletes queries recursively, gets the society information, and processes the first step operations for the get-all command.

public class QueriesByOriginatorAndTarget implements UnaryPredicate
Unary Predicate to query the blackboard for relays which have the given target and details drilldown query with the given originator, as the content.

public class QueriesByParent implements UnaryPredicate
Unary Predicate to query the blackboard for relays with DetailsDrillDownQuery with the given parent, as the content

public class QueriesComparator implements Comparator
Comparator used to compare two query relays.

public class QueriesPredicate implements UnaryPredicate
Unary Predicate to query the blackboard for relays with the given id and have AggregationDrillDownQuery or a DetailsDrillDownQuery or a CompleteEvents as the content. If the id is null all such relays are returned.

public class Query
This class stores the information about a users query. The object of this class is mapped to a user.

public class QueryManagerPlugin extends ComponentPlugin
This plugin receives and processes the commands from the clients. Uses commands tool to process the commands.

public class QueryResultsPlugin extends ComponentPlugin
This Plugin returns the results of the user queries as and when available. When a user requests for all the results for all of his/her queries, this plugin returns all the results starting from second level to the lowest available level, one by one.

public class ResultsTool
Results Tool is used by the QueryResultsPlugin to send the results of the queries to the appropriate user. Given a query results, the tool looks for the owner of the query and sent the results to that owner.

public class RootQueriesPredicate implements UnaryPredicate
Unary Predicate to look for the relays with AggregationDrillDown query as content.

public class UserInfo
This class provides a map for users to the queries.

SCAggregationResultSet.java
This class extends AggregationResultSet to add to add newgetAddedAtoms() method.

4.3.12 Seccon QueryManager Package Classes

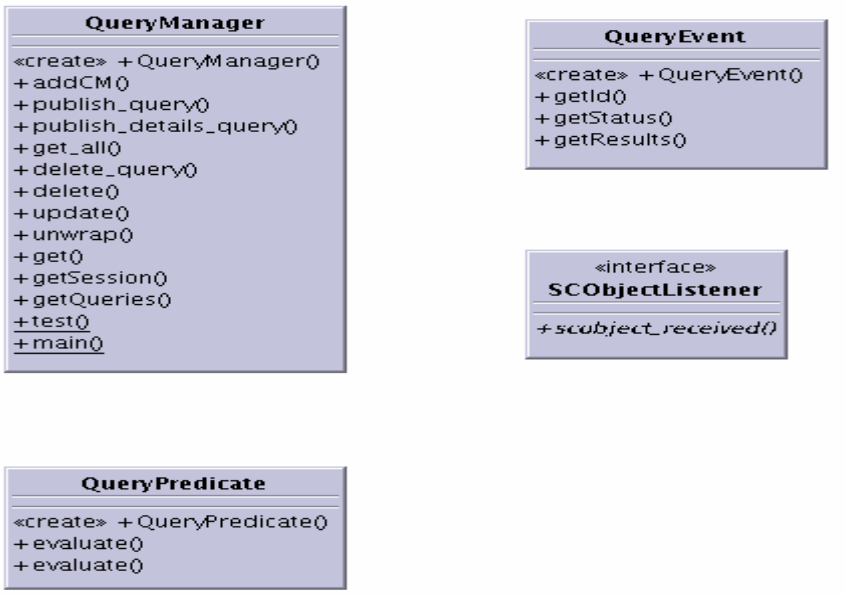


Figure 60: Classes in edu.memphis.issrl.seccon.querymanager classes

public class QueryEvent
Object with the information about the query result that was updated

public class QueryManager extends SCLayer
This class manages all the queries returned by the M&R manager. It is the interaction module between the communication module and the GUI

public class QueryPredicate
Java version of a Python predicate applied over a IDMEF Message

public interface SCObjectListener
Interface for receiving the updated query manager messages.

4.3.13 Seccon SnapInGui Package Classes

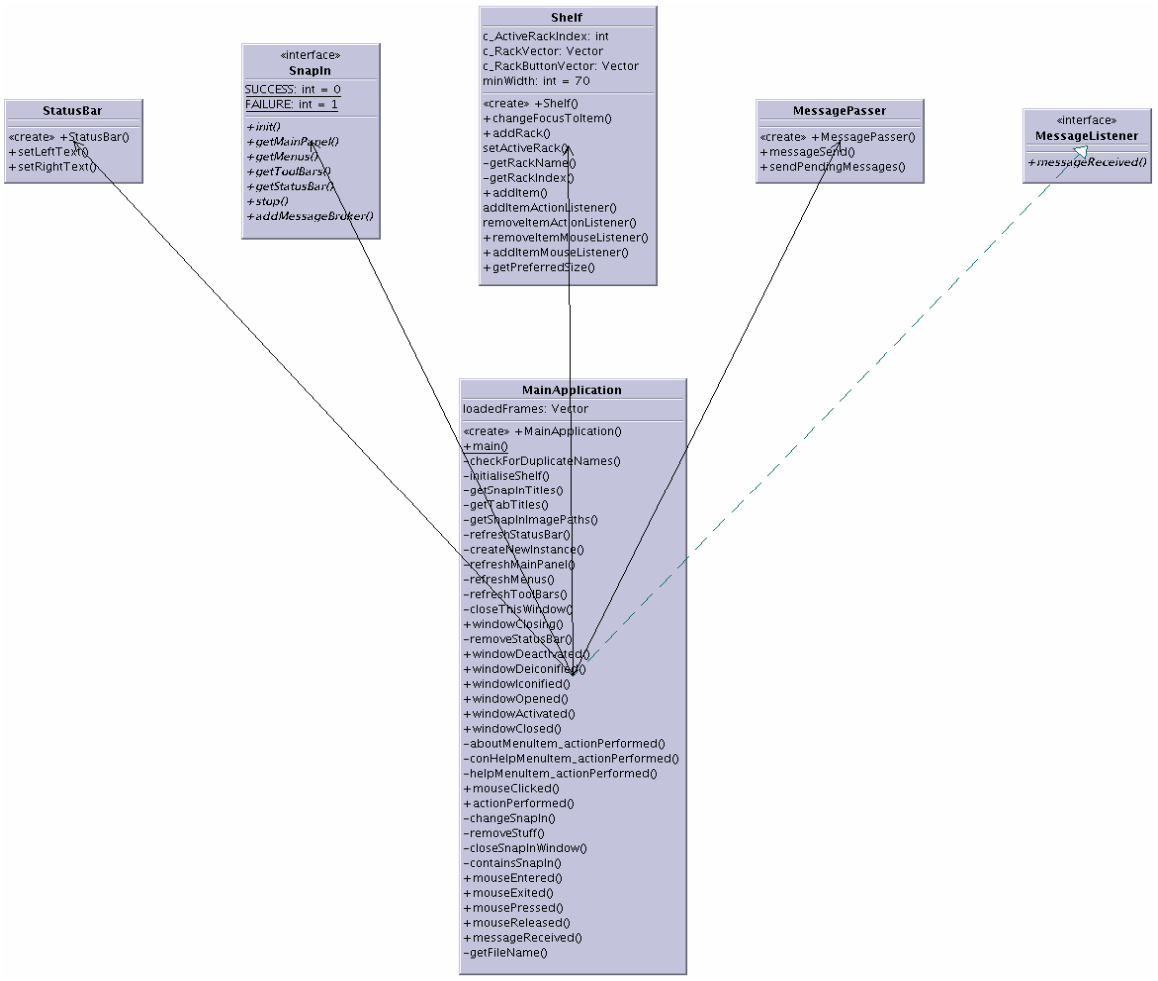


Figure 61: Classes in edu.memphis.issrl.snapingui package

public class ApplicationProperties

This class holds all the properties for the whole application. Mainly this holds the Colors of the Components. This contains methods to set the components background and foreground colors. This acts as a central color panel from which we can manipulate all the colors of the components. Each method in this class sets the corresponding panel color.

public class BorderlessButton extends JButton implements MouseListener
This button extends from JButton but doesn't have borders. Borders are not visible until the mouse is over the button.

public class ConfigFileReader
Reads from the file passed to its constructor file should contain KEY= VALUE is the separator for key and value.

public class Help extends JFrame implements HyperlinkListener
Class, which displays help in html format.

public class MainApplication extends JFrame implements ActionListener, WindowListener, MouseListener, MessageListener
This class is the main class for the Snap-In GUI. This contains Icons for each SnapIn (plugin).

public interface MessageBroker
Inteface, which transfers messages between SnapIns.

public class MessageEvent
Event, which is passed when a message is sent to SnapIns

public interface MessageListener
Listener interface, which listens for messages

public class MessagePasser implements MessageBroker

class Rack extends JPanel implements ActionListener
Rack represents a container to which Scroller is added.

class Scroller extends JScrollPane
Scroller represents a container to which actual buttons and labels are added. This also displays up arrow and down arrow buttons if the size of the scroller decreases.

public class Shelf extends JPanel
This is the component that holds the SnapIn icons. SnapIns can be categorised in to different categories. Each category is put in a different Rack.SnapIn click events are notified to a single listener. In the listener different SnapIns are identified using <code>getActionCommand<code> on source of the event, which returns the title of the SnapIn.

public interface SnapIn extends MessageListener
This class is the interface between SnapIn-GUI(MainApplication) and the Snap-Ins. Every Snap-In should implement this interface. Icon for the Snap-In is provided by SnapIn-GUI's config file.

public class SnapInWindow extends JFrame
This is the window in which a SnapIn is opened saperately.

public class StatusBar extends JPanel
This class is the status bar, which is displayed in the bottom.

public class UofMTableSorter extends DefaultTableModel implements TableModel Listener
Table sorter

4.3.14 Seccon Test Package Classes

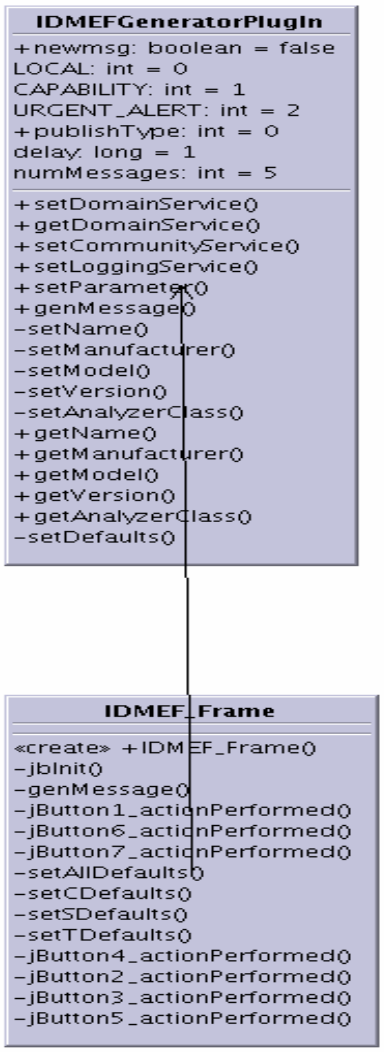


Figure 62: Classes edu.memphsi.issrl.test package

public class IDMEF_Frame extends JFrame
UI to let the user specify the attributes for idmef objects generated to test the security console.

public class IDMEFGeneratorPlugIn extends SensorPlugin implements SensorInfo
IDMEFGenerator generates IDMEF events at regular intervals. This PlugIn is used to test the functionality of the Security Console.

4.3.15 Seccon Xmining Package Classes

This package provides all the services required by the Security Console in order to read, write, update and erase xml messages in the eXist XML data base. The class ***MEDriver*** is responsible for establishing connection (open and close) with the data base (eXist). This ***MEDriver*** class constructor receives the name of a configuration file as parameter (***Mining.Properties***). The parameter URL in this class can point to an embedded eXist Data Base or to a stand alone eXist Data Base running in a local or remote machine.

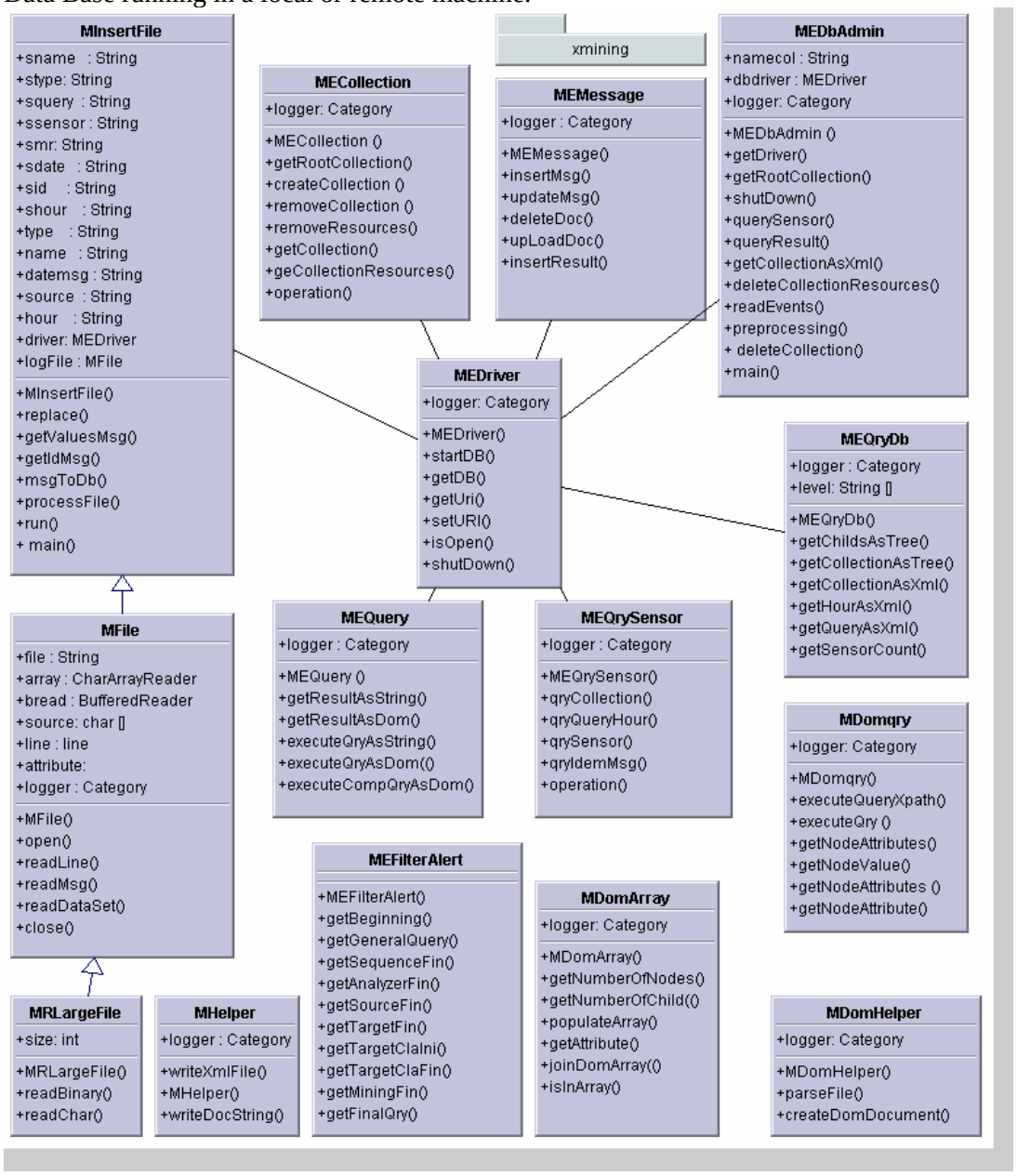


Figure 63: Classes edu.memphis.issrl.seccon.Xmining package

public class MRLargeFile extends Object

public class MFile extends Object

public class MInsertFile implements Runnable

This group of class is responsible for upload the logger.log (Security Console stores the IDEMF messages coming from the society in here) file to the eXist DB. The first class is responsible for read large files, the second one for read each IDEMF message and the last one is responsible for insert each message in the proper eXist collection.

public class MEDriver

This class is responsible for start up and shutdown an eXist DB.

public class MEColletion

This class is responsible for creates and removes collections, removes the xml messages inside a collection, retrieves the resources inside a collection and in general do all the tasks required by the Security Console in order to manage the collections inside the eXist DB.

public class MEMessage extends Object

This class is in charge of inserts, removes, updates and upload IDEMF messages in a eXist collection.

public class MEQrySensor

This class is in charge of develop an Xquery all over the collections inside the eXist DB. The collections in the eXist DB are similar to a directory structure in an operating system. The IDEMF messages coming from the society are store in the following path of collections: /db/Date/Hour/QueryId/SensorId/IDEMF.message.xml. Db is the root collection, date is the form yyyy-mm-dd. Hour is a string of two characters. QueryId represents the query name (from where the message is coming) and finally IDEMFmessage.xml represents each IDEMF message coming from the society.

public class MEQuery

This class is responsible of perform a Xquery over a eXist collection. It has the ability to return the result such as string or document.

public class MEQryDb

This class can retrieve in a DefaultMutableTreeNode a tree that represent the structure of collection inside the eXist DB. This class has also the ability to retrieve the tree structure of the eXist DB as an org.w3c.dom document.

public class MEDbAdmin

This class is called by other classes in other packages in order to perform tasks such as delete a collection, sent a query to the eXist DB, shutdown the DB among other. This class uses the other class in this package in order to provide these services.

public class MEFilterAlert extends Object

This class contains the Xqueries required by the Security Console in order to gets the result from the eXist DB. It has xqueries for the statistical GUI as well as the mining algorithms.

public class MDomqry extends Object

This class is responsible of perform a xpath query over an org.w3.dom document according with the xpath specification.

public class MDomArray extends Object

This class is responsible for "join" two objects: A string object type [] [] and an org.w3.dom document. It retrieves all the nodes inside ob the object [] [] from the doc document.

public class MHelper extends Object

public class MDom extends Object

These classes are responsible for provide services related with org.w3.dom objects such as create a document, parse an xml file, parse a xml string and so on.

4.3.16 Seccon Xmining Episodes Package Classes

The classes in this package implement two algorithms. The first one is called Discovery of frequent episodes in event sequences written by H. Mannila, H. Toivonen and I. Verkano, report c-1997-15, The University of Helsinki, Finland. The second one is called Fast algorithms for mining association rules written by R. Agrawal and R. Srikant, IBM Almaden Research Center, Proc. 20th Int. Conf. Very Large Data Bases, VLDB, 1994. The idea is first get serial and parallel frequent episodes and then uses these frequent episodes in order to get the association rules.

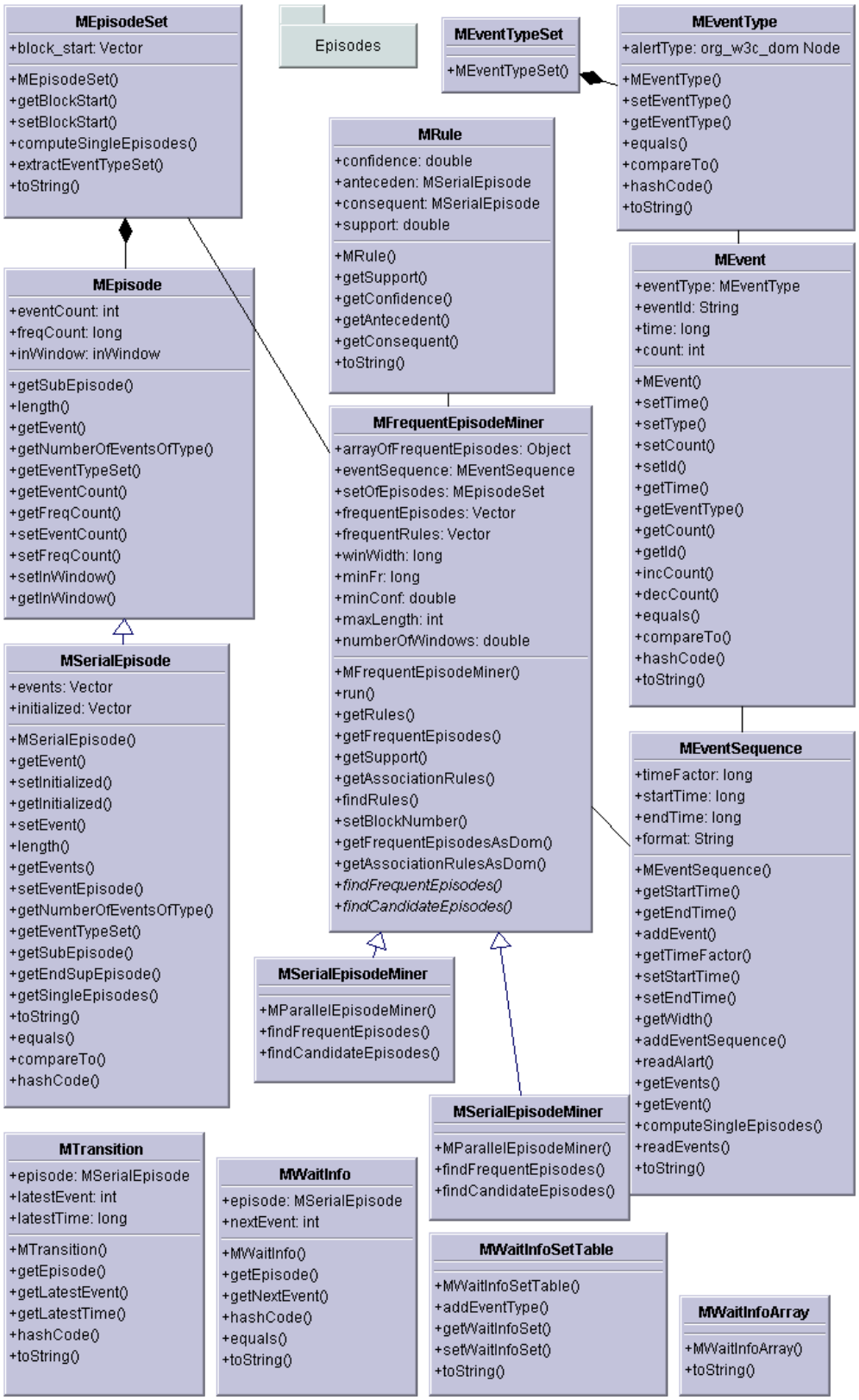


Figure 64: Classes edu.memphis.issr.seccon.Xmining.Episodes package

The classes inside this package are the following:

public class MEventType

This class defines one alert type. In this case an alert type is defined such as orgw3.node object. It uses the fields Source/Address/address, Target/Address/address and Classification/name from an IDEMF message. With these fields the frequent episodes tell the Security Console user, which are the attackers, the attacks type and the victims. This class implements method such as hashcode, equal and comparator that allow the algorithm to compare different event types.

public class MEventTypeSet extends HashSet

This class contains a group of object type MEventType.

public class MEvent extends Object

This class represents one alert in the algorithm discovery of frequent episodes in event sequences. This class is made up with the attributes event type, time of the alert and alert ID.

public class MEventSequence extends Hashtable

This class represents the input for the algorithm Discovery of frequent episodes in event sequences. This class extends HashTable in order to speed up the time to access each event inside the class. The start time represents the first alert in the sequence and end time represent the latest time.

public abstract class MEpisode

One episode represents a group of events occurring frequently together.

public class MSerialEpisode extends MEpisode implements Comparable

This class represents a serial and parallel episode according with the requirement in the paper discover of frequent episodes in event sequences. One episode is a group of events that happen frequently together.

public class MEpisodeSet extends TreeSet

Represent a group of episodes.

public abstract class MFrequentEpisodeMiner implements Runnable

In class implement the algorithms described in the paper discover of frequent episodes in event sequences. This class implements the algorithms number one and two (see the details in the paper). The Classes that extends this class must implement find frequent episodes and find candidate episodes algorithms in order to discover parallel and serial episodes. Also this class implements the algorithm fast algorithm for mining association rules.

public class MParallelEpisodeMiner extends MFrequentEpisodeMiner

This class implements the algorithms number 3 and four described in the paper discover of frequent episodes. The algorithm number three find candidate's episodes and the algorithm number 4 count how many times a parallel episode happen.

public class MSerialEpisodeMiner extends MFrequentEpisodeMiner

This class implements the algorithms number 3 and five described in the paper discover of frequent episodes. The algorithm number three find candidate's episodes and the algorithm number 4 count how many times a serial episode happen.

public class MTransition

This class is used by the object MSerialEpisodeMiner algorithm that finds frequent episodes. This class store serial episodes, the latest time the episode occurred and the latest event that have happened.

public class MWaitInfoSetTable extends Hashtable

This class is used by the object MSerialEpisodeMiner algorithm that finds frequent episodes. The structure waits describe the requirements for this class. See the details in the paper Discover frequent episodes in event sequence – algorithm number 5.

public class MWaitInfoArray extends ArrayList

This class is used by the object MSerialEpisodeMiner algorithm that finds frequent episodes. Each object inside of the object MWaitInfoSetTable is an array list.

public class MWaitInfo

Each one of the objects in the arraylist MWaitInfoArray is an object MWaitInfo.